

ChibiOS/HAL

6.0.1

Reference Manual

Tue May 1 2018 09:42:14

Contents

1	ChibiOS/HAL	1
1.1	Copyright	1
1.2	Introduction	1
1.3	Related Documents	1
2	Deprecated List	3
3	Module Index	5
3.1	Modules	5
4	Hierarchical Index	7
4.1	Class Hierarchy	7
5	Data Structure Index	11
5.1	Data Structures	11
6	File Index	17
6.1	File List	17
7	Module Documentation	23
7.1	Generic Accelerometer Interface	23
7.1.1	Detailed Description	23
7.1.2	Macro Definition Documentation	24
7.1.2.1	<code>_base_accelerometer_methods_alone</code>	24
7.1.2.2	<code>_base_accelerometer_methods</code>	24
7.1.2.3	<code>_base_accelerometer_data</code>	24
7.1.2.4	<code>accelerometerGetAxesNumber</code>	24
7.1.2.5	<code>accelerometerReadRaw</code>	24
7.1.2.6	<code>accelerometerReadCooked</code>	25
7.1.2.7	<code>accelerometerSetBias</code>	25
7.1.2.8	<code>accelerometerResetBias</code>	26
7.1.2.9	<code>accelerometerSetSensitivity</code>	26
7.1.2.10	<code>accelerometerResetSensitivity</code>	27
7.2	ADC Driver	28

7.2.1	Detailed Description	28
7.2.2	Driver State Machine	28
7.2.3	ADC Operations	28
7.2.3.1	ADC Conversion Groups	28
7.2.3.2	ADC Conversion Modes	29
7.2.3.3	ADC Callbacks	29
7.2.4	Macro Definition Documentation	31
7.2.4.1	ADC_USE_WAIT	31
7.2.4.2	ADC_USE_MUTUAL_EXCLUSION	31
7.2.4.3	_adc_reset_i	31
7.2.4.4	_adc_reset_s	32
7.2.4.5	_adc_wakeup_isr	32
7.2.4.6	_adc_timeout_isr	32
7.2.4.7	_adc_isr_half_code	33
7.2.4.8	_adc_isr_full_code	33
7.2.4.9	_adc_isr_error_code	33
7.2.4.10	PLATFORM_ADC_USE_ADC1	34
7.2.5	Typedef Documentation	34
7.2.5.1	adcsample_t	34
7.2.5.2	adc_channels_num_t	34
7.2.5.3	ADCDriver	35
7.2.5.4	adccallback_t	35
7.2.5.5	adcerrorcallback_t	35
7.2.6	Enumeration Type Documentation	35
7.2.6.1	adcstate_t	35
7.2.6.2	adcerror_t	35
7.2.7	Function Documentation	36
7.2.7.1	adcInit(void)	36
7.2.7.2	adcObjectInit(ADCDriver *adcp)	36
7.2.7.3	adcStart(ADCDriver *adcp, const ADCConfig *config)	36
7.2.7.4	adcStop(ADCDriver *adcp)	37
7.2.7.5	adcStartConversion(ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)	38
7.2.7.6	adcStartConversionI(ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)	39
7.2.7.7	adcStopConversion(ADCDriver *adcp)	39
7.2.7.8	adcStopConversionI(ADCDriver *adcp)	40
7.2.7.9	adcConvert(ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)	40
7.2.7.10	adcAcquireBus(ADCDriver *adcp)	41
7.2.7.11	adcReleaseBus(ADCDriver *adcp)	42

7.2.7.12	<code>adc_lld_init(void)</code>	43
7.2.7.13	<code>adc_lld_start(ADCDriver *adcp)</code>	43
7.2.7.14	<code>adc_lld_stop(ADCDriver *adcp)</code>	43
7.2.7.15	<code>adc_lld_start_conversion(ADCDriver *adcp)</code>	43
7.2.7.16	<code>adc_lld_stop_conversion(ADCDriver *adcp)</code>	44
7.2.8	Variable Documentation	44
7.2.8.1	<code>ADCD1</code>	44
7.3	Generic Barometer Interface	45
7.3.1	Detailed Description	45
7.3.2	Macro Definition Documentation	45
7.3.2.1	<code>_base_barometer_methods_alone</code>	45
7.3.2.2	<code>_base_barometer_methods</code>	46
7.3.2.3	<code>_base_barometer_data</code>	46
7.3.2.4	<code>barometerGetChannelsNumber</code>	46
7.3.2.5	<code>barometerReadRaw</code>	46
7.3.2.6	<code>barometerReadCooked</code>	47
7.3.2.7	<code>barometerSetBias</code>	47
7.3.2.8	<code>barometerResetBias</code>	48
7.3.2.9	<code>barometerSetSensitivity</code>	48
7.3.2.10	<code>barometerResetSensitivity</code>	49
7.4	CAN Driver	50
7.4.1	Detailed Description	50
7.4.2	Driver State Machine	50
7.4.3	Macro Definition Documentation	53
7.4.3.1	<code>CAN_LIMIT_WARNING</code>	53
7.4.3.2	<code>CAN_LIMIT_ERROR</code>	53
7.4.3.3	<code>CAN_BUS_OFF_ERROR</code>	53
7.4.3.4	<code>CAN_FRAMING_ERROR</code>	53
7.4.3.5	<code>CAN_OVERFLOW_ERROR</code>	53
7.4.3.6	<code>CAN_ANY_MAILBOX</code>	53
7.4.3.7	<code>CAN_USE_SLEEP_MODE</code>	53
7.4.3.8	<code>CAN_ENFORCE_USE_CALLBACKS</code>	53
7.4.3.9	<code>CAN_MAILBOX_TO_MASK</code>	53
7.4.3.10	<code>canTransmit</code>	54
7.4.3.11	<code>canReceive</code>	54
7.4.3.12	<code>_can_tx_empty_isr</code>	54
7.4.3.13	<code>_can_rx_full_isr</code>	54
7.4.3.14	<code>_can_wakeup_isr</code>	54
7.4.3.15	<code>_can_error_isr</code>	55
7.4.3.16	<code>CAN_TX_MAILBOXES</code>	55

7.4.3.17	CAN_RX_MAILBOXES	55
7.4.3.18	PLATFORM_CAN_USE_CAN1	55
7.4.4	Typedef Documentation	55
7.4.4.1	CANDriver	55
7.4.4.2	canmbx_t	55
7.4.4.3	can_callback_t	55
7.4.5	Enumeration Type Documentation	55
7.4.5.1	canstate_t	56
7.4.6	Function Documentation	56
7.4.6.1	canInit(void)	56
7.4.6.2	canObjectInit(CANDriver *canp)	56
7.4.6.3	canStart(CANDriver *canp, const CANConfig *config)	57
7.4.6.4	canStop(CANDriver *canp)	57
7.4.6.5	canTryTransmit(CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)	58
7.4.6.6	canTryReceive(CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)	59
7.4.6.7	canTransmitTimeout(CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, sysinterval_t timeout)	60
7.4.6.8	canReceiveTimeout(CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, sysinterval_t timeout)	61
7.4.6.9	canSleep(CANDriver *canp)	62
7.4.6.10	canWakeup(CANDriver *canp)	63
7.4.6.11	can_lld_init(void)	64
7.4.6.12	can_lld_start(CANDriver *canp)	64
7.4.6.13	can_lld_stop(CANDriver *canp)	65
7.4.6.14	can_lld_is_tx_empty(CANDriver *canp, canmbx_t mailbox)	65
7.4.6.15	can_lld_transmit(CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)	65
7.4.6.16	can_lld_is_rx_nonempty(CANDriver *canp, canmbx_t mailbox)	66
7.4.6.17	can_lld_receive(CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)	66
7.4.6.18	can_lld_sleep(CANDriver *canp)	66
7.4.6.19	can_lld_wakeup(CANDriver *canp)	67
7.4.7	Variable Documentation	67
7.4.7.1	CAND1	67
7.5	Generic Compass Interface	68
7.5.1	Detailed Description	68
7.5.2	Macro Definition Documentation	68
7.5.2.1	_base_compass_methods_alone	68
7.5.2.2	_base_compass_methods	69
7.5.2.3	_base_compass_data	69
7.5.2.4	compassGetAxesNumber	69
7.5.2.5	compassReadRaw	69

7.5.2.6	compassReadCooked	70
7.5.2.7	compassSetBias	70
7.5.2.8	compassResetBias	71
7.5.2.9	compassSetSensitivity	71
7.5.2.10	compassResetSensitivity	72
7.6	Crypto Driver	73
7.6.1	Detailed Description	73
7.6.2	Macro Definition Documentation	77
7.6.2.1	HAL_CRY_MAX_KEY_SIZE	77
7.6.2.2	HAL_CRY_USE_FALLBACK	77
7.6.2.3	HAL_CRY_USE_FALLBACK	77
7.6.2.4	HAL_CRY_ENFORCE_FALLBACK	78
7.6.2.5	PLATFORM_CRY_USE_CRY1	78
7.6.3	Typedef Documentation	78
7.6.3.1	bitsize_t	78
7.6.3.2	crykey_t	78
7.6.3.3	CRYDriver	78
7.6.4	Enumeration Type Documentation	78
7.6.4.1	crystate_t	78
7.6.4.2	cryerror_t	79
7.6.4.3	cryalgorithm_t	79
7.6.5	Function Documentation	79
7.6.5.1	cryInit(void)	79
7.6.5.2	cryObjectInit(CRYDriver *cryp)	79
7.6.5.3	cryStart(CRYDriver *cryp, const CRYConfig *config)	80
7.6.5.4	cryStop(CRYDriver *cryp)	80
7.6.5.5	cryLoadTransientKey(CRYDriver *cryp, cryalgorithm_t algorithm, size_t size, const uint8_t *keyp)	81
7.6.5.6	cryEncryptAES(CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)	82
7.6.5.7	cryDecryptAES(CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)	83
7.6.5.8	cryEncryptAES_ECB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)	84
7.6.5.9	cryDecryptAES_ECB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)	85
7.6.5.10	cryEncryptAES_CBC(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)	86
7.6.5.11	cryDecryptAES_CBC(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)	87
7.6.5.12	cryEncryptAES_CFB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)	88
7.6.5.13	cryDecryptAES_CFB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)	89

7.6.5.14	<code>cryEncryptAES_CTR(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	90
7.6.5.15	<code>cryDecryptAES_CTR(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	91
7.6.5.16	<code>cryEncryptAES_GCM(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv, size_t aadsize, const uint8_t *aad, uint8_t *authtag)</code>	92
7.6.5.17	<code>cryDecryptAES_GCM(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv, size_t aadsize, const uint8_t *aad, uint8_t *authtag)</code>	93
7.6.5.18	<code>cryEncryptDES(CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)</code>	94
7.6.5.19	<code>cryDecryptDES(CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)</code>	95
7.6.5.20	<code>cryEncryptDES_ECB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)</code>	96
7.6.5.21	<code>cryDecryptDES_ECB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)</code>	97
7.6.5.22	<code>cryEncryptDES_CBC(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	98
7.6.5.23	<code>cryDecryptDES_CBC(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	99
7.6.5.24	<code>crySHA1Init(CRYDriver *cryp, SHA1Context *sha1ctxp)</code>	100
7.6.5.25	<code>crySHA1Update(CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)</code>	101
7.6.5.26	<code>crySHA1Final(CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)</code>	101
7.6.5.27	<code>crySHA256Init(CRYDriver *cryp, SHA256Context *sha256ctxp)</code>	102
7.6.5.28	<code>crySHA256Update(CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)</code>	103
7.6.5.29	<code>crySHA256Final(CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)</code>	104
7.6.5.30	<code>crySHA512Init(CRYDriver *cryp, SHA512Context *sha512ctxp)</code>	104
7.6.5.31	<code>crySHA512Update(CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)</code>	105
7.6.5.32	<code>crySHA512Final(CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)</code>	106
7.6.5.33	<code>cryTRNG(CRYDriver *cryp, uint8_t *out)</code>	107
7.6.5.34	<code>cry_lld_init(void)</code>	107
7.6.5.35	<code>cry_lld_start(CRYDriver *cryp)</code>	107
7.6.5.36	<code>cry_lld_stop(CRYDriver *cryp)</code>	108
7.6.5.37	<code>cry_lld_loadkey(CRYDriver *cryp, cryalgorithm_t algorithm, size_t size, const uint8_t *keyp)</code>	108
7.6.5.38	<code>cry_lld_encrypt_AES(CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)</code>	108
7.6.5.39	<code>cry_lld_decrypt_AES(CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)</code>	109
7.6.5.40	<code>cry_lld_encrypt_AES_ECB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)</code>	110

7.6.5.41	<code>cry_llt_decrypt_AES_ECB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)</code>	110
7.6.5.42	<code>cry_llt_encrypt_AES_CBC(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	111
7.6.5.43	<code>cry_llt_decrypt_AES_CBC(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	112
7.6.5.44	<code>cry_llt_encrypt_AES_CFB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	112
7.6.5.45	<code>cry_llt_decrypt_AES_CFB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	113
7.6.5.46	<code>cry_llt_encrypt_AES_CTR(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	114
7.6.5.47	<code>cry_llt_decrypt_AES_CTR(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	115
7.6.5.48	<code>cry_llt_encrypt_AES_GCM(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv, size_t aadsize, const uint8_t *aad, uint8_t *authtag)</code>	115
7.6.5.49	<code>cry_llt_decrypt_AES_GCM(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv, size_t aadsize, const uint8_t *aad, uint8_t *authtag)</code>	116
7.6.5.50	<code>cry_llt_encrypt_DES(CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)</code>	117
7.6.5.51	<code>cry_llt_decrypt_DES(CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)</code>	117
7.6.5.52	<code>cry_llt_encrypt_DES_ECB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)</code>	118
7.6.5.53	<code>cry_llt_decrypt_DES_ECB(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)</code>	119
7.6.5.54	<code>cry_llt_encrypt_DES_CBC(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	119
7.6.5.55	<code>cry_llt_decrypt_DES_CBC(CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)</code>	120
7.6.5.56	<code>cry_llt_SHA1_init(CRYDriver *cryp, SHA1Context *sha1ctxp)</code>	121
7.6.5.57	<code>cry_llt_SHA1_update(CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)</code>	121
7.6.5.58	<code>cry_llt_SHA1_final(CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)</code>	122
7.6.5.59	<code>cry_llt_SHA256_init(CRYDriver *cryp, SHA256Context *sha256ctxp)</code>	122
7.6.5.60	<code>cry_llt_SHA256_update(CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)</code>	123
7.6.5.61	<code>cry_llt_SHA256_final(CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)</code>	123
7.6.5.62	<code>cry_llt_SHA512_init(CRYDriver *cryp, SHA512Context *sha512ctxp)</code>	124
7.6.5.63	<code>cry_llt_SHA512_update(CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)</code>	124
7.6.5.64	<code>cry_llt_SHA512_final(CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)</code>	125
7.6.5.65	<code>cry_llt_TRNG(CRYDriver *cryp, uint8_t *out)</code>	125

7.6.6	Variable Documentation	126
7.6.6.1	CRYD1	126
7.7	DAC Driver	127
7.7.1	Detailed Description	127
7.7.2	Macro Definition Documentation	129
7.7.2.1	DAC_USE_WAIT	129
7.7.2.2	DAC_USE_MUTUAL_EXCLUSION	129
7.7.2.3	_dac_wait_s	129
7.7.2.4	_dac_reset_i	130
7.7.2.5	_dac_reset_s	130
7.7.2.6	_dac_wakeup_isr	130
7.7.2.7	_dac_timeout_isr	131
7.7.2.8	_dac_isr_half_code	131
7.7.2.9	_dac_isr_full_code	131
7.7.2.10	_dac_isr_error_code	132
7.7.2.11	DAC_MAX_CHANNELS	133
7.7.2.12	PLATFORM_DAC_USE_DAC1	133
7.7.3	Typedef Documentation	133
7.7.3.1	dacchannel_t	133
7.7.3.2	DACDriver	133
7.7.3.3	dacsample_t	133
7.7.3.4	daccallback_t	133
7.7.3.5	dacerrorcallback_t	133
7.7.4	Enumeration Type Documentation	134
7.7.4.1	dacstate_t	134
7.7.4.2	dacerror_t	134
7.7.5	Function Documentation	134
7.7.5.1	dacInit(void)	134
7.7.5.2	dacObjectInit(DACDriver *daccp)	135
7.7.5.3	dacStart(DACDriver *daccp, const DACConfig *config)	135
7.7.5.4	dacStop(DACDriver *daccp)	136
7.7.5.5	dacPutChannelX(DACDriver *daccp, dacchannel_t channel, dacsample_t sample)	137
7.7.5.6	dacStartConversion(DACDriver *daccp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)	137
7.7.5.7	dacStartConversionI(DACDriver *daccp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)	138
7.7.5.8	dacStopConversion(DACDriver *daccp)	139
7.7.5.9	dacStopConversionI(DACDriver *daccp)	139
7.7.5.10	dacConvert(DACDriver *daccp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)	140
7.7.5.11	dacAcquireBus(DACDriver *daccp)	141

7.7.5.12	<code>dacReleaseBus(DACDriver *dACP)</code>	141
7.7.5.13	<code>dacIld_init(void)</code>	142
7.7.5.14	<code>dacIld_start(DACDriver *dACP)</code>	142
7.7.5.15	<code>dacIld_stop(DACDriver *dACP)</code>	143
7.7.5.16	<code>dacIld_put_channel(DACDriver *dACP, dacchannel_t channel, dacsample_t sample)</code>	143
7.7.5.17	<code>dacIld_start_conversion(DACDriver *dACP)</code>	143
7.7.5.18	<code>dacIld_stop_conversion(DACDriver *dACP)</code>	144
7.7.6	Variable Documentation	144
7.7.6.1	DACD1	144
7.8	EXT Driver	145
7.8.1	Detailed Description	145
7.8.2	Driver State Machine	145
7.8.3	EXT Operations.	145
7.8.4	Macro Definition Documentation	147
7.8.4.1	EXT_CH_MODE_EDGES_MASK	147
7.8.4.2	EXT_CH_MODE_DISABLED	147
7.8.4.3	EXT_CH_MODE_RISING_EDGE	147
7.8.4.4	EXT_CH_MODE_FALLING_EDGE	147
7.8.4.5	EXT_CH_MODE_BOTH_EDGES	148
7.8.4.6	EXT_CH_MODE_LOW_LEVEL	148
7.8.4.7	EXT_CH_MODE_AUTOSTART	148
7.8.4.8	<code>extChannelEnable</code>	148
7.8.4.9	<code>extChannelDisable</code>	148
7.8.4.10	<code>extSetChannelMode</code>	148
7.8.4.11	EXT_MAX_CHANNELS	149
7.8.4.12	PLATFORM_EXT_USE_EXT1	149
7.8.5	Typedef Documentation	149
7.8.5.1	EXTDriver	149
7.8.5.2	<code>expchannel_t</code>	149
7.8.5.3	<code>extcallback_t</code>	149
7.8.6	Enumeration Type Documentation	149
7.8.6.1	<code>extstate_t</code>	150
7.8.7	Function Documentation	150
7.8.7.1	<code>extInit(void)</code>	150
7.8.7.2	<code>extObjectInit(EXTDriver *extp)</code>	150
7.8.7.3	<code>extStart(EXTDriver *extp, const EXTConfig *config)</code>	150
7.8.7.4	<code>extStop(EXTDriver *extp)</code>	151
7.8.7.5	<code>extChannelEnable(EXTDriver *extp, expchannel_t channel)</code>	152
7.8.7.6	<code>extChannelDisable(EXTDriver *extp, expchannel_t channel)</code>	152

7.8.7.7	extSetChannelModel(EXTDriver *extp, expchannel_t channel, const EXT↔ ChannelConfig *extcp)	153
7.8.7.8	ext_lld_init(void)	154
7.8.7.9	ext_lld_start(EXTDriver *extp)	154
7.8.7.10	ext_lld_stop(EXTDriver *extp)	154
7.8.7.11	ext_lld_channel_enable(EXTDriver *extp, expchannel_t channel)	155
7.8.7.12	ext_lld_channel_disable(EXTDriver *extp, expchannel_t channel)	155
7.8.8	Variable Documentation	155
7.8.8.1	EXTD1	155
7.9	Generic Flash Driver Interface	156
7.9.1	Detailed Description	156
7.9.2	Driver State Machine	156
7.9.3	Macro Definition Documentation	158
7.9.3.1	_base_flash_methods_alone	158
7.9.3.2	_base_flash_methods	158
7.9.3.3	_base_flash_data	158
7.9.3.4	getBaseFlash	158
7.9.3.5	flashGetDescriptor	158
7.9.3.6	flashRead	159
7.9.3.7	flashProgram	159
7.9.3.8	flashStartEraseAll	160
7.9.3.9	flashStartEraseSector	160
7.9.3.10	flashQueryErase	161
7.9.3.11	flashVerifyErase	161
7.9.4	Typedef Documentation	162
7.9.4.1	flash_offset_t	162
7.9.4.2	flash_sector_t	162
7.9.5	Enumeration Type Documentation	162
7.9.5.1	flash_state_t	162
7.9.5.2	flash_error_t	162
7.9.6	Function Documentation	162
7.9.6.1	flashWaitErase(BaseFlash *devp)	162
7.9.6.2	flashGetSectorOffset(BaseFlash *devp, flash_sector_t sector)	163
7.9.6.3	flashGetSectorSize(BaseFlash *devp, flash_sector_t sector)	163
7.10	GPT Driver	164
7.10.1	Detailed Description	164
7.10.2	Driver State Machine	164
7.10.3	GPT Operations.	164
7.10.4	Macro Definition Documentation	167
7.10.4.1	gptChangeIntervall	167

7.10.4.2	gptGetIntervalX	167
7.10.4.3	gptGetCounterX	167
7.10.4.4	PLATFORM_GPT_USE_GPT1	168
7.10.4.5	gpt_lld_change_interval	168
7.10.5	Typedef Documentation	169
7.10.5.1	GPTDriver	169
7.10.5.2	gptcallback_t	169
7.10.5.3	gptfreq_t	169
7.10.5.4	gptcnt_t	169
7.10.6	Enumeration Type Documentation	169
7.10.6.1	gptstate_t	169
7.10.7	Function Documentation	169
7.10.7.1	gptInit(void)	169
7.10.7.2	gptObjectInit(GPTDriver *gptp)	170
7.10.7.3	gptStart(GPTDriver *gptp, const GPTConfig *config)	170
7.10.7.4	gptStop(GPTDriver *gptp)	171
7.10.7.5	gptChangeInterval(GPTDriver *gptp, gptcnt_t interval)	171
7.10.7.6	gptStartContinuous(GPTDriver *gptp, gptcnt_t interval)	172
7.10.7.7	gptStartContinuousI(GPTDriver *gptp, gptcnt_t interval)	173
7.10.7.8	gptStartOneShot(GPTDriver *gptp, gptcnt_t interval)	173
7.10.7.9	gptStartOneShotI(GPTDriver *gptp, gptcnt_t interval)	174
7.10.7.10	gptStopTimer(GPTDriver *gptp)	174
7.10.7.11	gptStopTimerI(GPTDriver *gptp)	175
7.10.7.12	gptPolledDelay(GPTDriver *gptp, gptcnt_t interval)	175
7.10.7.13	gpt_lld_init(void)	176
7.10.7.14	gpt_lld_start(GPTDriver *gptp)	176
7.10.7.15	gpt_lld_stop(GPTDriver *gptp)	177
7.10.7.16	gpt_lld_start_timer(GPTDriver *gptp, gptcnt_t interval)	177
7.10.7.17	gpt_lld_stop_timer(GPTDriver *gptp)	177
7.10.7.18	gpt_lld_polled_delay(GPTDriver *gptp, gptcnt_t interval)	177
7.10.8	Variable Documentation	178
7.10.8.1	GPTD1	178
7.11	Generic Gyroscope Interface	179
7.11.1	Detailed Description	179
7.11.2	Macro Definition Documentation	179
7.11.2.1	_base_gyroscope_methods_alone	179
7.11.2.2	_base_gyroscope_methods	180
7.11.2.3	_base_gyroscope_data	180
7.11.2.4	gyroscopeGetAxesNumber	180
7.11.2.5	gyroscopeReadRaw	180

7.11.2.6	gyroscopeReadCooked	181
7.11.2.7	gyroscopeSampleBias	181
7.11.2.8	gyroscopeSetBias	182
7.11.2.9	gyroscopeResetBias	182
7.11.2.10	gyroscopeSetSensitivity	183
7.11.2.11	gyroscopeResetSensitivity	183
7.12	HAL Driver	184
7.12.1	Detailed Description	184
7.12.2	Macro Definition Documentation	185
7.12.2.1	_CHIBIOS_HAL_	185
7.12.2.2	CH_HAL_STABLE	185
7.12.2.3	HAL_VERSION	185
7.12.2.4	CH_HAL_MAJOR	185
7.12.2.5	CH_HAL_MINOR	185
7.12.2.6	CH_HAL_PATCH	185
7.12.3	Function Documentation	185
7.12.3.1	halInit(void)	185
7.12.3.2	hal_lld_init(void)	186
7.13	I/O Buffers Queues	187
7.13.1	Detailed Description	187
7.13.2	Macro Definition Documentation	189
7.13.2.1	BQ_BUFFER_SIZE	189
7.13.2.2	bqSizeX	189
7.13.2.3	bqSpacel	189
7.13.2.4	bqGetLinkX	189
7.13.2.5	bqIsSuspendedX	190
7.13.2.6	bqSuspendI	190
7.13.2.7	bqResumeX	191
7.13.2.8	ibqIsEmptyI	191
7.13.2.9	ibqIsFullI	192
7.13.2.10	obqIsEmptyI	192
7.13.2.11	obqIsFullI	193
7.13.3	Typedef Documentation	193
7.13.3.1	io_buffers_queue_t	193
7.13.3.2	bqnotify_t	193
7.13.3.3	input_buffers_queue_t	193
7.13.3.4	output_buffers_queue_t	193
7.13.4	Function Documentation	194
7.13.4.1	ibqObjectInit(input_buffers_queue_t *ibqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t infy, void *link)	194

7.13.4.2	ibqResetI(input_buffers_queue_t *ibqp)	194
7.13.4.3	ibqGetEmptyBufferI(input_buffers_queue_t *ibqp)	195
7.13.4.4	ibqPostFullBufferI(input_buffers_queue_t *ibqp, size_t size)	195
7.13.4.5	ibqGetFullBufferTimeout(input_buffers_queue_t *ibqp, sysinterval_t timeout)	196
7.13.4.6	ibqGetFullBufferTimeoutS(input_buffers_queue_t *ibqp, sysinterval_t timeout)	197
7.13.4.7	ibqReleaseEmptyBuffer(input_buffers_queue_t *ibqp)	198
7.13.4.8	ibqReleaseEmptyBufferS(input_buffers_queue_t *ibqp)	198
7.13.4.9	ibqGetTimeout(input_buffers_queue_t *ibqp, sysinterval_t timeout)	199
7.13.4.10	ibqReadTimeout(input_buffers_queue_t *ibqp, uint8_t *bp, size_t n, sysinterval_t timeout)	200
7.13.4.11	obqObjectInit(output_buffers_queue_t *obqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t only, void *link)	201
7.13.4.12	obqResetI(output_buffers_queue_t *obqp)	201
7.13.4.13	obqGetFullBufferI(output_buffers_queue_t *obqp, size_t *sizep)	202
7.13.4.14	obqReleaseEmptyBufferI(output_buffers_queue_t *obqp)	202
7.13.4.15	obqGetEmptyBufferTimeout(output_buffers_queue_t *obqp, sysinterval_t timeout)	203
7.13.4.16	obqGetEmptyBufferTimeoutS(output_buffers_queue_t *obqp, sysinterval_t timeout)	204
7.13.4.17	obqPostFullBuffer(output_buffers_queue_t *obqp, size_t size)	205
7.13.4.18	obqPostFullBufferS(output_buffers_queue_t *obqp, size_t size)	205
7.13.4.19	obqPutTimeout(output_buffers_queue_t *obqp, uint8_t b, sysinterval_t timeout)	206
7.13.4.20	obqWriteTimeout(output_buffers_queue_t *obqp, const uint8_t *bp, size_t n, sysinterval_t timeout)	207
7.13.4.21	obqTryFlushI(output_buffers_queue_t *obqp)	208
7.13.4.22	obqFlush(output_buffers_queue_t *obqp)	208
7.14	Abstract I/O Channel	210
7.14.1	Detailed Description	210
7.14.2	Macro Definition Documentation	211
7.14.2.1	_base_channel_methods	211
7.14.2.2	_base_channel_data	211
7.14.2.3	chnPutTimeout	212
7.14.2.4	chnGetTimeout	212
7.14.2.5	chnWrite	213
7.14.2.6	chnWriteTimeout	213
7.14.2.7	chnRead	213
7.14.2.8	chnReadTimeout	215
7.14.2.9	chnControl	215
7.14.2.10	CHN_NO_ERROR	216
7.14.2.11	CHN_CONNECTED	216
7.14.2.12	CHN_DISCONNECTED	216
7.14.2.13	CHN_INPUT_AVAILABLE	216

7.14.2.14	CHN_OUTPUT_EMPTY	216
7.14.2.15	CHN_TRANSMISSION_END	216
7.14.2.16	_base_asynchronous_channel_methods	216
7.14.2.17	_base_asynchronous_channel_data	216
7.14.2.18	chnGetEventSource	217
7.14.2.19	chnAddFlagsI	218
7.15	Abstract Files	219
7.15.1	Detailed Description	219
7.15.2	Macro Definition Documentation	220
7.15.2.1	FILE_OK	220
7.15.2.2	FILE_ERROR	220
7.15.2.3	FILE_EOF	220
7.15.2.4	_file_stream_methods	220
7.15.2.5	_file_stream_data	220
7.15.2.6	fileStreamWrite	220
7.15.2.7	fileStreamRead	221
7.15.2.8	fileStreamPut	221
7.15.2.9	fileStreamGet	222
7.15.2.10	fileStreamClose	222
7.15.2.11	fileStreamGetError	223
7.15.2.12	fileStreamGetSize	223
7.15.2.13	fileStreamGetPosition	224
7.15.2.14	fileStreamSeek	224
7.15.3	Typedef Documentation	224
7.15.3.1	fileoffset_t	224
7.16	Abstract I/O Block Device	225
7.16.1	Detailed Description	225
7.16.2	Driver State Machine	225
7.16.3	Macro Definition Documentation	226
7.16.3.1	_base_block_device_methods	226
7.16.3.2	_base_block_device_data	226
7.16.3.3	blkGetDriverState	227
7.16.3.4	blkIsTransferring	227
7.16.3.5	blkIsInserted	228
7.16.3.6	blkIsWriteProtected	228
7.16.3.7	blkConnect	228
7.16.3.8	blkDisconnect	229
7.16.3.9	blkRead	229
7.16.3.10	blkWrite	230
7.16.3.11	blkSync	230

7.16.3.12 blkGetInfo	231
7.16.4 Enumeration Type Documentation	231
7.16.4.1 blkstate_t	231
7.17 Base Object	232
7.17.1 Detailed Description	232
7.17.2 Macro Definition Documentation	232
7.17.2.1 _base_object_methods	232
7.17.2.2 _base_object_data	232
7.17.2.3 objGetInstance	233
7.18 Abstract Peripheral Interfaces	235
7.18.1 Detailed Description	235
7.19 I/O Bytes Queues	236
7.19.1 Detailed Description	236
7.19.2 Macro Definition Documentation	238
7.19.2.1 Q_OK	238
7.19.2.2 Q_TIMEOUT	238
7.19.2.3 Q_RESET	238
7.19.2.4 Q_EMPTY	238
7.19.2.5 Q_FULL	238
7.19.2.6 qSizeX	238
7.19.2.7 qSpacel	239
7.19.2.8 qGetLink	239
7.19.2.9 iqGetFullI	239
7.19.2.10 iqGetEmptyI	240
7.19.2.11 iqlsEmptyI	240
7.19.2.12 iqlsFullI	241
7.19.2.13 iqGet	241
7.19.2.14 oqGetFullI	242
7.19.2.15 oqGetEmptyI	242
7.19.2.16 oqlsEmptyI	242
7.19.2.17 oqlsFullI	243
7.19.2.18 oqPut	243
7.19.3 Typedef Documentation	244
7.19.3.1 io_queue_t	244
7.19.3.2 qnotify_t	244
7.19.3.3 input_queue_t	244
7.19.3.4 output_queue_t	244
7.19.4 Function Documentation	244
7.19.4.1 iq_read(input_queue_t *iqp, uint8_t *bp, size_t n)	244
7.19.4.2 oq_write(output_queue_t *oqp, const uint8_t *bp, size_t n)	245

7.19.4.3	iqObjectInit(input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)	245
7.19.4.4	iqResetI(input_queue_t *iqp)	246
7.19.4.5	iqPutI(input_queue_t *iqp, uint8_t b)	246
7.19.4.6	iqGetI(input_queue_t *iqp)	247
7.19.4.7	iqGetTimeout(input_queue_t *iqp, sysinterval_t timeout)	248
7.19.4.8	iqReadI(input_queue_t *iqp, uint8_t *bp, size_t n)	249
7.19.4.9	iqReadTimeout(input_queue_t *iqp, uint8_t *bp, size_t n, sysinterval_t timeout)	249
7.19.4.10	oqObjectInit(output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)	250
7.19.4.11	oqResetI(output_queue_t *oqp)	251
7.19.4.12	oqPutI(output_queue_t *oqp, uint8_t b)	251
7.19.4.13	oqPutTimeout(output_queue_t *oqp, uint8_t b, sysinterval_t timeout)	252
7.19.4.14	oqGetI(output_queue_t *oqp)	253
7.19.4.15	oqWriteI(output_queue_t *oqp, const uint8_t *bp, size_t n)	254
7.19.4.16	oqWriteTimeout(output_queue_t *oqp, const uint8_t *bp, size_t n, sysinterval_t timeout)	254
7.20	Abstract Streams	256
7.20.1	Detailed Description	256
7.20.2	Macro Definition Documentation	256
7.20.2.1	_base_sequential_stream_methods	256
7.20.2.2	_base_sequential_stream_data	257
7.20.2.3	streamWrite	257
7.20.2.4	streamRead	257
7.20.2.5	streamPut	258
7.20.2.6	streamGet	258
7.21	Generic Hygrometer Interface	259
7.21.1	Detailed Description	259
7.21.2	Macro Definition Documentation	259
7.21.2.1	_base_hygrometer_methods_alone	259
7.21.2.2	_base_hygrometer_methods	260
7.21.2.3	_base_hygrometer_data	260
7.21.2.4	hygrometerGetChannelsNumber	260
7.21.2.5	hygrometerReadRaw	260
7.21.2.6	hygrometerReadCooked	261
7.21.2.7	hygrometerSetBias	261
7.21.2.8	hygrometerResetBias	262
7.21.2.9	hygrometerSetSensitivity	262
7.21.2.10	hygrometerResetSensitivity	263
7.22	I2C Driver	264
7.22.1	Detailed Description	264

7.22.2	Driver State Machine	264
7.22.3	Macro Definition Documentation	266
7.22.3.1	I2C_NO_ERROR	266
7.22.3.2	I2C_BUS_ERROR	266
7.22.3.3	I2C_ARBITRATION_LOST	267
7.22.3.4	I2C_ACK_FAILURE	267
7.22.3.5	I2C_OVERRUN	267
7.22.3.6	I2C_PEC_ERROR	267
7.22.3.7	I2C_TIMEOUT	267
7.22.3.8	I2C_SMB_ALERT	267
7.22.3.9	I2C_USE_MUTUAL_EXCLUSION	267
7.22.3.10	_i2c_wakeup_isr	267
7.22.3.11	_i2c_wakeup_error_isr	267
7.22.3.12	i2cMasterTransmit	268
7.22.3.13	i2cMasterReceive	268
7.22.3.14	PLATFORM_I2C_USE_I2C1	268
7.22.3.15	i2c_llc_get_errors	268
7.22.4	Typedef Documentation	269
7.22.4.1	i2caddr_t	269
7.22.4.2	i2cflags_t	269
7.22.4.3	I2CDriver	269
7.22.5	Enumeration Type Documentation	269
7.22.5.1	i2cstate_t	269
7.22.6	Function Documentation	269
7.22.6.1	i2cInit(void)	269
7.22.6.2	i2cObjectInit(I2CDriver *i2cp)	270
7.22.6.3	i2cStart(I2CDriver *i2cp, const I2CConfig *config)	270
7.22.6.4	i2cStop(I2CDriver *i2cp)	271
7.22.6.5	i2cGetErrors(I2CDriver *i2cp)	271
7.22.6.6	i2cMasterTransmitTimeout(I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)	272
7.22.6.7	i2cMasterReceiveTimeout(I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)	273
7.22.6.8	i2cAcquireBus(I2CDriver *i2cp)	274
7.22.6.9	i2cReleaseBus(I2CDriver *i2cp)	274
7.22.6.10	i2c_llc_init(void)	275
7.22.6.11	i2c_llc_start(I2CDriver *i2cp)	275
7.22.6.12	i2c_llc_stop(I2CDriver *i2cp)	276
7.22.6.13	i2c_llc_master_receive_timeout(I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)	276

7.22.6.14	i2c_lld_master_transmit_timeout(I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)	276
7.22.7	Variable Documentation	277
7.22.7.1	I2CD1	277
7.23	I2S Driver	278
7.23.1	Detailed Description	278
7.23.2	Driver State Machine	278
7.23.3	Macro Definition Documentation	279
7.23.3.1	i2sStartExchangeI	279
7.23.3.2	i2sStopExchangeI	280
7.23.3.3	_i2s_isr_half_code	280
7.23.3.4	_i2s_isr_full_code	280
7.23.3.5	PLATFORM_I2S_USE_I2S1	281
7.23.4	Typedef Documentation	281
7.23.4.1	I2SDriver	281
7.23.4.2	i2scallback_t	281
7.23.5	Enumeration Type Documentation	282
7.23.5.1	i2sstate_t	282
7.23.6	Function Documentation	282
7.23.6.1	i2sInit(void)	282
7.23.6.2	i2sObjectInit(I2SDriver *i2sp)	282
7.23.6.3	i2sStart(I2SDriver *i2sp, const I2SConfig *config)	283
7.23.6.4	i2sStop(I2SDriver *i2sp)	283
7.23.6.5	i2sStartExchange(I2SDriver *i2sp)	284
7.23.6.6	i2sStopExchange(I2SDriver *i2sp)	284
7.23.6.7	i2s_lld_init(void)	285
7.23.6.8	i2s_lld_start(I2SDriver *i2sp)	285
7.23.6.9	i2s_lld_stop(I2SDriver *i2sp)	285
7.23.6.10	i2s_lld_start_exchange(I2SDriver *i2sp)	286
7.23.6.11	i2s_lld_stop_exchange(I2SDriver *i2sp)	286
7.23.7	Variable Documentation	286
7.23.7.1	I2SD1	286
7.24	ICU Driver	287
7.24.1	Detailed Description	287
7.24.2	Driver State Machine	287
7.24.3	ICU Operations	287
7.24.4	Macro Definition Documentation	290
7.24.4.1	icuStartCaptureI	290
7.24.4.2	icuStopCaptureI	290
7.24.4.3	icuEnableNotificationsI	291

7.24.4.4	icuDisableNotificationsI	291
7.24.4.5	icuAreNotificationsEnabledX	291
7.24.4.6	icuGetWidthX	292
7.24.4.7	icuGetPeriodX	292
7.24.4.8	_icu_isr_invoke_width_cb	293
7.24.4.9	_icu_isr_invoke_period_cb	293
7.24.4.10	_icu_isr_invoke_overflow_cb	293
7.24.4.11	PLATFORM_ICU_USE_ICU1	294
7.24.4.12	icu_lld_get_width	294
7.24.4.13	icu_lld_get_period	294
7.24.4.14	icu_lld_are_notifications_enabled	295
7.24.5	Typedef Documentation	295
7.24.5.1	ICUDriver	295
7.24.5.2	icucallback_t	295
7.24.5.3	icufreq_t	295
7.24.5.4	icucnt_t	295
7.24.6	Enumeration Type Documentation	296
7.24.6.1	icustate_t	296
7.24.6.2	icumode_t	296
7.24.7	Function Documentation	296
7.24.7.1	icuInit(void)	296
7.24.7.2	icuObjectInit(ICUDriver *icup)	296
7.24.7.3	icuStart(ICUDriver *icup, const ICUConfig *config)	297
7.24.7.4	icuStop(ICUDriver *icup)	297
7.24.7.5	icuStartCapture(ICUDriver *icup)	298
7.24.7.6	icuWaitCapture(ICUDriver *icup)	298
7.24.7.7	icuStopCapture(ICUDriver *icup)	299
7.24.7.8	icuEnableNotifications(ICUDriver *icup)	300
7.24.7.9	icuDisableNotifications(ICUDriver *icup)	301
7.24.7.10	icu_lld_init(void)	301
7.24.7.11	icu_lld_start(ICUDriver *icup)	302
7.24.7.12	icu_lld_stop(ICUDriver *icup)	302
7.24.7.13	icu_lld_start_capture(ICUDriver *icup)	302
7.24.7.14	icu_lld_wait_capture(ICUDriver *icup)	302
7.24.7.15	icu_lld_stop_capture(ICUDriver *icup)	303
7.24.7.16	icu_lld_enable_notifications(ICUDriver *icup)	303
7.24.7.17	icu_lld_disable_notifications(ICUDriver *icup)	303
7.24.8	Variable Documentation	304
7.24.8.1	ICUD1	304
7.25	JESD216 Compliant Flash Driver Interface	305

7.25.1	Detailed Description	305
7.25.2	Macro Definition Documentation	306
7.25.2.1	JESD216_BUS_MODE	306
7.25.2.2	JESD216_SHARED_BUS	306
7.25.2.3	_jesd216_flash_methods_alone	306
7.25.2.4	_jesd216_flash_methods	306
7.25.2.5	_jesd216_flash_data	306
7.26	MAC Driver	307
7.26.1	Detailed Description	307
7.26.2	Macro Definition Documentation	309
7.26.2.1	MAC_USE_ZERO_COPY	309
7.26.2.2	MAC_USE_EVENTS	309
7.26.2.3	macGetReceiveEventSource	309
7.26.2.4	macWriteTransmitDescriptor	309
7.26.2.5	macReadReceiveDescriptor	310
7.26.2.6	macGetNextTransmitBuffer	310
7.26.2.7	macGetNextReceiveBuffer	311
7.26.2.8	MAC_SUPPORTS_ZERO_COPY	311
7.26.2.9	PLATFORM_MAC_USE_MAC1	311
7.26.3	Typedef Documentation	311
7.26.3.1	MACDriver	311
7.26.4	Enumeration Type Documentation	311
7.26.4.1	macstate_t	311
7.26.5	Function Documentation	312
7.26.5.1	macInit(void)	312
7.26.5.2	macObjectInit(MACDriver *macp)	312
7.26.5.3	macStart(MACDriver *macp, const MACConfig *config)	313
7.26.5.4	macStop(MACDriver *macp)	313
7.26.5.5	macWaitTransmitDescriptor(MACDriver *macp, MACTransmitDescriptor *tdp, sysinterval_t timeout)	314
7.26.5.6	macReleaseTransmitDescriptor(MACTransmitDescriptor *tdp)	315
7.26.5.7	macWaitReceiveDescriptor(MACDriver *macp, MACReceiveDescriptor *rdp, sysinterval_t timeout)	315
7.26.5.8	macReleaseReceiveDescriptor(MACReceiveDescriptor *rdp)	316
7.26.5.9	macPollLinkStatus(MACDriver *macp)	317
7.26.5.10	mac_lld_init(void)	317
7.26.5.11	mac_lld_start(MACDriver *macp)	318
7.26.5.12	mac_lld_stop(MACDriver *macp)	318
7.26.5.13	mac_lld_get_transmit_descriptor(MACDriver *macp, MACTransmitDescriptor *tdp)	318
7.26.5.14	mac_lld_release_transmit_descriptor(MACTransmitDescriptor *tdp)	319

7.26.5.15	mac_lld_get_receive_descriptor(MACDriver *macp, MACReceiveDescriptor *rdp)	319
7.26.5.16	mac_lld_release_receive_descriptor(MACReceiveDescriptor *rdp)	319
7.26.5.17	mac_lld_poll_link_status(MACDriver *macp)	320
7.26.5.18	mac_lld_write_transmit_descriptor(MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)	320
7.26.5.19	mac_lld_read_receive_descriptor(MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)	321
7.26.5.20	mac_lld_get_next_transmit_buffer(MACTransmitDescriptor *tdp, size_t size, size_t *sizep)	321
7.26.5.21	mac_lld_get_next_receive_buffer(MACReceiveDescriptor *rdp, size_t *sizep)	321
7.26.6	Variable Documentation	322
7.26.6.1	ETHD1	322
7.27	HAL	323
7.27.1	Detailed Description	323
7.27.2	HAL Device Drivers Architecture	323
7.27.3	HAL Normal Device Drivers	323
7.27.3.1	Diagram	324
7.27.4	HAL Complex Device Drivers	324
7.27.5	HAL Interfaces	324
7.27.6	HAL Inner Code	324
7.28	Configuration	326
7.28.1	Detailed Description	326
7.28.2	Macro Definition Documentation	328
7.28.2.1	HAL_USE_PAL	328
7.28.2.2	HAL_USE_ADC	328
7.28.2.3	HAL_USE_CAN	328
7.28.2.4	HAL_USE_CRY	328
7.28.2.5	HAL_USE_DAC	328
7.28.2.6	HAL_USE_EXT	328
7.28.2.7	HAL_USE_GPT	328
7.28.2.8	HAL_USE_I2C	329
7.28.2.9	HAL_USE_I2S	329
7.28.2.10	HAL_USE_ICU	329
7.28.2.11	HAL_USE_MAC	329
7.28.2.12	HAL_USE_MMC_SPI	329
7.28.2.13	HAL_USE_PWM	329
7.28.2.14	HAL_USE_QSPI	329
7.28.2.15	HAL_USE_RTC	329
7.28.2.16	HAL_USE_SDC	329
7.28.2.17	HAL_USE_SERIAL	329
7.28.2.18	HAL_USE_SERIAL_USB	329

7.28.2.19 HAL_USE_SPI	329
7.28.2.20 HAL_USE_UART	330
7.28.2.21 HAL_USE_USB	330
7.28.2.22 HAL_USE_WDG	330
7.28.2.23 PAL_USE_CALLBACKS	330
7.28.2.24 PAL_USE_WAIT	330
7.28.2.25 ADC_USE_WAIT	330
7.28.2.26 ADC_USE_MUTUAL_EXCLUSION	330
7.28.2.27 CAN_USE_SLEEP_MODE	330
7.28.2.28 CAN_ENFORCE_USE_CALLBACKS	330
7.28.2.29 HAL_CRY_USE_FALLBACK	331
7.28.2.30 HAL_CRY_ENFORCE_FALLBACK	331
7.28.2.31 DAC_USE_WAIT	331
7.28.2.32 DAC_USE_MUTUAL_EXCLUSION	331
7.28.2.33 I2C_USE_MUTUAL_EXCLUSION	331
7.28.2.34 MAC_USE_ZERO_COPY	331
7.28.2.35 MAC_USE_EVENTS	331
7.28.2.36 MMC_NICE_WAITING	331
7.28.2.37 QSPI_USE_WAIT	332
7.28.2.38 QSPI_USE_MUTUAL_EXCLUSION	332
7.28.2.39 SDC_INIT_RETRY	332
7.28.2.40 SDC_MMC_SUPPORT	332
7.28.2.41 SDC_NICE_WAITING	332
7.28.2.42 SDC_INIT_OCR_V20	332
7.28.2.43 SDC_INIT_OCR	332
7.28.2.44 SERIAL_DEFAULT_BITRATE	332
7.28.2.45 SERIAL_BUFFERS_SIZE	333
7.28.2.46 SERIAL_USB_BUFFERS_SIZE	333
7.28.2.47 SERIAL_USB_BUFFERS_NUMBER	333
7.28.2.48 SPI_USE_WAIT	333
7.28.2.49 SPI_USE_CIRCULAR	333
7.28.2.50 SPI_USE_MUTUAL_EXCLUSION	333
7.28.2.51 SPI_SELECT_MODE	334
7.28.2.52 UART_USE_WAIT	334
7.28.2.53 UART_USE_MUTUAL_EXCLUSION	334
7.28.2.54 USB_USE_WAIT	334
7.29 Normal Drivers	335
7.29.1 Detailed Description	335
7.30 Complex Drivers	337
7.30.1 Detailed Description	337

7.31	Interfaces and Classes	338
7.31.1	Detailed Description	338
7.32	Inner Code	339
7.32.1	Detailed Description	339
7.33	Support Code	340
7.33.1	Detailed Description	340
7.34	OSAL	341
7.34.1	Detailed Description	341
7.34.2	Macro Definition Documentation	346
7.34.2.1	OSAL_ST_RESOLUTION	346
7.34.2.2	OSAL_ST_FREQUENCY	346
7.34.2.3	OSAL_ST_MODE	346
7.34.2.4	OSAL_IRQ_PRIORITY_LEVELS	346
7.34.2.5	OSAL_IRQ_MAXIMUM_PRIORITY	346
7.34.2.6	OSAL_DBG_ENABLE_ASSERTS	346
7.34.2.7	OSAL_DBG_ENABLE_CHECKS	346
7.34.2.8	osalDbgAssert	347
7.34.2.9	osalDbgCheck	347
7.34.2.10	osalDbgCheckClassI	348
7.34.2.11	osalDbgCheckClassS	348
7.34.2.12	OSAL_IRQ_IS_VALID_PRIORITY	348
7.34.2.13	OSAL_IRQ_PROLOGUE	348
7.34.2.14	OSAL_IRQ_EPILOGUE	348
7.34.2.15	OSAL_IRQ_HANDLER	348
7.34.2.16	OSAL_S2I	348
7.34.2.17	OSAL_MS2I	349
7.34.2.18	OSAL_US2I	349
7.34.2.19	OSAL_S2RTC	350
7.34.2.20	OSAL_MS2RTC	350
7.34.2.21	OSAL_US2RTC	351
7.34.2.22	osalThreadSleepSeconds	351
7.34.2.23	osalThreadSleepMilliseconds	351
7.34.2.24	osalThreadSleepMicroseconds	352
7.34.3	Typedef Documentation	352
7.34.3.1	syssts_t	352
7.34.3.2	msg_t	352
7.34.3.3	sys_time_t	352
7.34.3.4	sysinterval_t	352
7.34.3.5	rtcnt_t	352
7.34.3.6	thread_reference_t	353

7.34.3.7	eventflags_t	353
7.34.3.8	event_source_t	353
7.34.3.9	eventcallback_t	353
7.34.3.10	mutex_t	353
7.34.4	Function Documentation	353
7.34.4.1	osalInit(void)	353
7.34.4.2	osalSysHalt(const char *reason)	353
7.34.4.3	osalSysPolledDelayX(rtcnt_t cycles)	354
7.34.4.4	osalOsTimerHandlerI(void)	354
7.34.4.5	osalOsRescheduleS(void)	354
7.34.4.6	osalOsGetSystemTimeX(void)	355
7.34.4.7	osalThreadSleepS(sysinterval_t time)	355
7.34.4.8	osalThreadSleep(sysinterval_t time)	355
7.34.4.9	osalThreadSuspendS(thread_reference_t *trp)	356
7.34.4.10	osalThreadSuspendTimeoutS(thread_reference_t *trp, sysinterval_t timeout)	356
7.34.4.11	osalThreadResumeI(thread_reference_t *trp, msg_t msg)	357
7.34.4.12	osalThreadResumeS(thread_reference_t *trp, msg_t msg)	357
7.34.4.13	osalThreadEnqueueTimeoutS(threads_queue_t *tqp, sysinterval_t timeout)	357
7.34.4.14	osalThreadDequeueNextI(threads_queue_t *tqp, msg_t msg)	358
7.34.4.15	osalThreadDequeueAllI(threads_queue_t *tqp, msg_t msg)	358
7.34.4.16	osalEventBroadcastFlagsI(event_source_t *esp, eventflags_t flags)	358
7.34.4.17	osalEventBroadcastFlags(event_source_t *esp, eventflags_t flags)	359
7.34.4.18	osalEventSetCallback(event_source_t *esp, eventcallback_t cb, void *param)	359
7.34.4.19	osalMutexLock(mutex_t *mp)	360
7.34.4.20	osalMutexUnlock(mutex_t *mp)	360
7.34.4.21	osalSysDisable(void)	360
7.34.4.22	osalSysEnable(void)	360
7.34.4.23	osalSysLock(void)	361
7.34.4.24	osalSysUnlock(void)	361
7.34.4.25	osalSysLockFromISR(void)	361
7.34.4.26	osalSysUnlockFromISR(void)	361
7.34.4.27	osalSysGetStatusAndLockX(void)	362
7.34.4.28	osalSysRestoreStatusX(syssts_t sts)	362
7.34.4.29	osalTimeAddX(systime_t systime, sysinterval_t interval)	362
7.34.4.30	osalTimeDiffX(systime_t start, systime_t end)	363
7.34.4.31	osalTimeInRangeX(systime_t time, systime_t start, systime_t end)	363
7.34.4.32	osalThreadQueueObjectInit(threads_queue_t *tqp)	363
7.34.4.33	osalEventObjectInit(event_source_t *esp)	364
7.34.4.34	osalMutexObjectInit(mutex_t *mp)	364
7.34.5	Variable Documentation	364

7.34.5.1	osal_halt_msg	364
7.34.5.2	osal_halt_msg	364
7.35	Managed Flash Storage Driver	365
7.35.1	Detailed Description	365
7.35.2	Macro Definition Documentation	367
7.35.2.1	RET_ON_ERROR	367
7.35.2.2	MFS_CFG_MAX_RECORDS	367
7.35.2.3	MFS_CFG_MAX_REPAIR_ATTEMPTS	367
7.35.2.4	MFS_CFG_WRITE_VERIFY	367
7.35.2.5	MFS_CFG_STRONG_CHECKING	367
7.35.2.6	MFS_CFG_BUFFER_SIZE	367
7.35.3	Typedef Documentation	368
7.35.3.1	mfs_id_t	368
7.35.4	Enumeration Type Documentation	368
7.35.4.1	mfs_bank_t	368
7.35.4.2	mfs_state_t	368
7.35.4.3	mfs_error_t	368
7.35.4.4	mfs_bank_state_t	368
7.35.4.5	mfs_record_state_t	368
7.35.5	Function Documentation	368
7.35.5.1	mfs_flash_read(MFSDriver *mfsp, flash_offset_t offset, size_t n, uint8_t *rp)	368
7.35.5.2	mfs_flash_write(MFSDriver *mfsp, flash_offset_t offset, size_t n, const uint8_t *wp)	369
7.35.5.3	mfs_flash_copy(MFSDriver *mfsp, flash_offset_t doffset, flash_offset_t soffset, uint32_t n)	369
7.35.5.4	mfs_record_check(MFSDriver *mfsp, mfs_data_header_t *dhdrp, flash_offset_t offset, flash_offset_t limit, mfs_record_state_t *sts)	370
7.35.5.5	mfs_bank_erase(MFSDriver *mfsp, mfs_bank_t bank)	370
7.35.5.6	mfs_bank_verify_erase(MFSDriver *mfsp, mfs_bank_t bank)	371
7.35.5.7	mfs_bank_write_header(MFSDriver *mfsp, mfs_bank_t bank, uint32_t cnt)	371
7.35.5.8	mfs_bank_scan_records(MFSDriver *mfsp, mfs_bank_t bank, mfs_bank_state_t *statep)	372
7.35.5.9	mfs_bank_get_state(MFSDriver *mfsp, mfs_bank_t bank, mfs_bank_state_t *statep, uint32_t *cntp)	373
7.35.5.10	mfs_bank_mount(MFSDriver *mfsp, mfs_bank_t bank, mfs_bank_state_t *statep)	374
7.35.5.11	mfs_garbage_collect(MFSDriver *mfsp)	375
7.35.5.12	mfs_try_mount(MFSDriver *mfsp)	376
7.35.5.13	mfs_mount(MFSDriver *mfsp)	376
7.35.5.14	mfsObjectInit(MFSDriver *mfsp)	377
7.35.5.15	mfsStart(MFSDriver *mfsp, const MFSCfg *config)	377
7.35.5.16	mfsStop(MFSDriver *mfsp)	378
7.35.5.17	mfsErase(MFSDriver *mfsp)	378

7.35.5.18	mfsReadRecord(MFSDriver *mfsp, mfs_id_t id, size_t *np, uint8_t *buffer)	379
7.35.5.19	mfsWriteRecord(MFSDriver *mfsp, mfs_id_t id, size_t n, const uint8_t *buffer)	380
7.35.5.20	mfsEraseRecord(MFSDriver *mfsp, mfs_id_t id)	380
7.35.5.21	mfsPerformGarbageCollection(MFSDriver *mfsp)	381
7.36	MII/RMII Header	383
7.36.1	Detailed Description	383
7.36.2	Macro Definition Documentation	385
7.36.2.1	MII_BMCR	385
7.36.2.2	MII_BMSR	385
7.36.2.3	MII_PHYSID1	385
7.36.2.4	MII_PHYSID2	385
7.36.2.5	MII_ADVERTISE	385
7.36.2.6	MII_LPA	386
7.36.2.7	MII_EXPANSION	386
7.36.2.8	MII_ANNPTR	386
7.36.2.9	MII_CTRL1000	386
7.36.2.10	MII_STAT1000	386
7.36.2.11	MII_ESTATUS	386
7.36.2.12	MII_PHYSTS	386
7.36.2.13	MII_MICR	386
7.36.2.14	MII_DCOUNTER	386
7.36.2.15	MII_FCSCOUNTER	386
7.36.2.16	MII_NWAYTEST	386
7.36.2.17	MII_RERRCOUNTER	386
7.36.2.18	MII_SREVISION	387
7.36.2.19	MII_RESV1	387
7.36.2.20	MII_LBRERROR	387
7.36.2.21	MII_PHYADDR	387
7.36.2.22	MII_RESV2	387
7.36.2.23	MII_TPISTATUS	387
7.36.2.24	MII_NCONFIG	387
7.36.2.25	BMCR_RESV	387
7.36.2.26	BMCR_CTST	387
7.36.2.27	BMCR_FULLDPLX	387
7.36.2.28	BMCR_ANRESTART	387
7.36.2.29	BMCR_ISOLATE	387
7.36.2.30	BMCR_PDOWN	388
7.36.2.31	BMCR_ANENABLE	388
7.36.2.32	BMCR_SPEED100	388
7.36.2.33	BMCR_LOOPBACK	388

7.36.2.34 BMCR_RESET	388
7.36.2.35 BMSR_ERCAP	388
7.36.2.36 BMSR_JCD	388
7.36.2.37 BMSR_LSTATUS	388
7.36.2.38 BMSR_ANEGCAPABLE	388
7.36.2.39 BMSR_RFAULT	388
7.36.2.40 BMSR_ANEGCOMPLETE	388
7.36.2.41 BMSR_MFPRESUPPCAP	388
7.36.2.42 BMSR_RESV	389
7.36.2.43 BMSR_10HALF	389
7.36.2.44 BMSR_10FULL	389
7.36.2.45 BMSR_100HALF	389
7.36.2.46 BMSR_100FULL	389
7.36.2.47 BMSR_100BASE4	389
7.36.2.48 ADVERTISE_SLCT	389
7.36.2.49 ADVERTISE_CSMA	389
7.36.2.50 ADVERTISE_10HALF	389
7.36.2.51 ADVERTISE_10FULL	389
7.36.2.52 ADVERTISE_100HALF	389
7.36.2.53 ADVERTISE_100FULL	389
7.36.2.54 ADVERTISE_100BASE4	390
7.36.2.55 ADVERTISE_PAUSE_CAP	390
7.36.2.56 ADVERTISE_PAUSE_ASYM	390
7.36.2.57 ADVERTISE_RESV	390
7.36.2.58 ADVERTISE_RFAULT	390
7.36.2.59 ADVERTISE_LPACK	390
7.36.2.60 ADVERTISE_NPAGE	390
7.36.2.61 LPA_SLCT	390
7.36.2.62 LPA_10HALF	390
7.36.2.63 LPA_10FULL	390
7.36.2.64 LPA_100HALF	390
7.36.2.65 LPA_100FULL	390
7.36.2.66 LPA_100BASE4	391
7.36.2.67 LPA_PAUSE_CAP	391
7.36.2.68 LPA_PAUSE_ASYM	391
7.36.2.69 LPA_RESV	391
7.36.2.70 LPA_RFAULT	391
7.36.2.71 LPA_LPACK	391
7.36.2.72 LPA_NPAGE	391
7.36.2.73 EXPANSION_NWAY	391

7.36.2.74 EXPANSION_LCWP	391
7.36.2.75 EXPANSION_ENABLENPAGE	391
7.36.2.76 EXPANSION_NPCAPABLE	391
7.36.2.77 EXPANSION_MFAULTS	391
7.36.2.78 EXPANSION_RESV	392
7.36.2.79 NWAYTEST_RESV1	392
7.36.2.80 NWAYTEST_LOOPBACK	392
7.36.2.81 NWAYTEST_RESV2	392
7.37 MMC over SPI Driver	393
7.37.1 Detailed Description	393
7.37.2 Driver State Machine	393
7.37.3 Driver Operations	393
7.37.4 Macro Definition Documentation	395
7.37.4.1 MMC_NICE_WAITING	395
7.37.4.2 _mmc_driver_methods	395
7.37.4.3 mmclsCardInserted	395
7.37.4.4 mmclsWriteProtected	395
7.37.5 Function Documentation	396
7.37.5.1 crc7(uint8_t crc, const uint8_t *buffer, size_t len)	396
7.37.5.2 wait(MMCDriver *mmcp)	396
7.37.5.3 send_hdr(MMCDriver *mmcp, uint8_t cmd, uint32_t arg)	397
7.37.5.4 recvr1(MMCDriver *mmcp)	397
7.37.5.5 recvr3(MMCDriver *mmcp, uint8_t *buffer)	398
7.37.5.6 send_command_R1(MMCDriver *mmcp, uint8_t cmd, uint32_t arg)	399
7.37.5.7 send_command_R3(MMCDriver *mmcp, uint8_t cmd, uint32_t arg, uint8_t *response)	400
7.37.5.8 read_CxD(MMCDriver *mmcp, uint8_t cmd, uint32_t cxd[4])	400
7.37.5.9 sync(MMCDriver *mmcp)	401
7.37.5.10 mmclnit(void)	402
7.37.5.11 mmcObjectInit(MMCDriver *mmcp)	402
7.37.5.12 mmcStart(MMCDriver *mmcp, const MMCCConfig *config)	402
7.37.5.13 mmcStop(MMCDriver *mmcp)	403
7.37.5.14 mmcConnect(MMCDriver *mmcp)	403
7.37.5.15 mmcDisconnect(MMCDriver *mmcp)	404
7.37.5.16 mmcStartSequentialRead(MMCDriver *mmcp, uint32_t startblk)	405
7.37.5.17 mmcSequentialRead(MMCDriver *mmcp, uint8_t *buffer)	406
7.37.5.18 mmcStopSequentialRead(MMCDriver *mmcp)	407
7.37.5.19 mmcStartSequentialWrite(MMCDriver *mmcp, uint32_t startblk)	408
7.37.5.20 mmcSequentialWrite(MMCDriver *mmcp, const uint8_t *buffer)	408
7.37.5.21 mmcStopSequentialWrite(MMCDriver *mmcp)	409

7.37.5.22	mmcSync(MMCDriver *mmcp)	410
7.37.5.23	mmcGetInfo(MMCDriver *mmcp, BlockDeviceInfo *bdip)	411
7.37.5.24	mmcErase(MMCDriver *mmcp, uint32_t startblk, uint32_t endblk)	411
7.37.6	Variable Documentation	412
7.37.6.1	mmc_vmt	412
7.37.6.2	crc7_lookup_table	412
7.38	MMC/SD Block Device	414
7.38.1	Detailed Description	414
7.38.2	Macro Definition Documentation	418
7.38.2.1	MMCSDBLOCK_SIZE	418
7.38.2.2	MMCSDBLOCK_R1_ERROR_MASK	418
7.38.2.3	MMCSDBLOCK_CMD8_PATTERN	418
7.38.2.4	MMCSDBLOCK_CSD_MMC_CSD_STRUCTURE_SLICE	418
7.38.2.5	MMCSDBLOCK_CID_SDC_CRC_SLICE	418
7.38.2.6	_mmcsd_block_device_methods	419
7.38.2.7	_mmcsd_block_device_data	419
7.38.2.8	MMCSDBLOCK_R1_ERROR	419
7.38.2.9	MMCSDBLOCK_R1_STS	419
7.38.2.10	MMCSDBLOCK_R1_IS_CARD_LOCKED	419
7.38.2.11	mmcsdGetCardCapacity	419
7.38.3	Function Documentation	420
7.38.3.1	_mmcsd_get_slice(const uint32_t *data, uint32_t end, uint32_t start)	420
7.38.3.2	_mmcsd_get_capacity(const uint32_t *csd)	420
7.38.3.3	_mmcsd_get_capacity_ext(const uint8_t *ext_csd)	421
7.38.3.4	_mmcsd_unpack_sdc_cid(const MMCSDBlockDevice *sdcp, unpacked_sdc↵ cid_t *cid_sdc)	421
7.38.3.5	_mmcsd_unpack_mmc_cid(const MMCSDBlockDevice *sdcp, unpacked_mmc↵ cid_t *cid_mmc)	422
7.38.3.6	_mmcsd_unpack_csd_mmc(const MMCSDBlockDevice *sdcp, unpacked↵ mmc_csd_t *csd_mmc)	422
7.38.3.7	_mmcsd_unpack_csd_v10(const MMCSDBlockDevice *sdcp, unpacked_sdc↵ csd_10_t *csd10)	423
7.38.3.8	_mmcsd_unpack_csd_v20(const MMCSDBlockDevice *sdcp, unpacked_sdc↵ csd_20_t *csd20)	423
7.39	PAL Driver	425
7.39.1	Detailed Description	425
7.39.2	Implementation Rules	425
7.39.2.1	Writing on input pads	425
7.39.2.2	Reading from output pads	425
7.39.2.3	Writing unused or unimplemented port bits	425
7.39.2.4	Reading from unused or unimplemented port bits	426

7.39.2.5	Reading or writing on pins associated to other functionalities	426
7.39.3	Macro Definition Documentation	431
7.39.3.1	PAL_MODE_RESET	431
7.39.3.2	PAL_MODE_UNCONNECTED	431
7.39.3.3	PAL_MODE_INPUT	431
7.39.3.4	PAL_MODE_INPUT_PULLUP	431
7.39.3.5	PAL_MODE_INPUT_PULLDOWN	431
7.39.3.6	PAL_MODE_INPUT_ANALOG	431
7.39.3.7	PAL_MODE_OUTPUT_PUSHPULL	431
7.39.3.8	PAL_MODE_OUTPUT_OPENDRAIN	431
7.39.3.9	PAL_LOW	432
7.39.3.10	PAL_HIGH	432
7.39.3.11	PAL_EVENT_MODE_EDGES_MASK	432
7.39.3.12	PAL_EVENT_MODE_DISABLED	432
7.39.3.13	PAL_EVENT_MODE_RISING_EDGE	432
7.39.3.14	PAL_EVENT_MODE_FALLING_EDGE	432
7.39.3.15	PAL_EVENT_MODE_BOTH_EDGES	432
7.39.3.16	PAL_USE_CALLBACKS	432
7.39.3.17	PAL_USE_WAIT	432
7.39.3.18	PAL_PORT_BIT	432
7.39.3.19	PAL_GROUP_MASK	433
7.39.3.20	_IOBUS_DATA	433
7.39.3.21	IOBUS_DECL	433
7.39.3.22	_pal_init_event	433
7.39.3.23	_pal_init_event	434
7.39.3.24	_pal_init_event	434
7.39.3.25	_pal_clear_event	435
7.39.3.26	_pal_clear_event	435
7.39.3.27	_pal_clear_event	435
7.39.3.28	_pal_isr_code	436
7.39.3.29	_pal_isr_code	436
7.39.3.30	_pal_isr_code	436
7.39.3.31	palInit	437
7.39.3.32	palReadPort	437
7.39.3.33	palReadLatch	438
7.39.3.34	palWritePort	438
7.39.3.35	palSetPort	438
7.39.3.36	palClearPort	439
7.39.3.37	palTogglePort	439
7.39.3.38	palReadGroup	439

7.39.3.39 palWriteGroup	440
7.39.3.40 palSetGroupMode	440
7.39.3.41 palReadPad	441
7.39.3.42 palWritePad	441
7.39.3.43 palSetPad	442
7.39.3.44 palClearPad	442
7.39.3.45 palTogglePad	443
7.39.3.46 palSetPadMode	443
7.39.3.47 palReadLine	443
7.39.3.48 palWriteLine	444
7.39.3.49 palSetLine	444
7.39.3.50 palClearLine	445
7.39.3.51 palToggleLine	445
7.39.3.52 palSetLineMode	445
7.39.3.53 palEnablePadEventI	446
7.39.3.54 palDisablePadEventI	446
7.39.3.55 palEnablePadEvent	446
7.39.3.56 palDisablePadEvent	447
7.39.3.57 palEnableLineEventI	447
7.39.3.58 palDisableLineEventI	448
7.39.3.59 palEnableLineEvent	448
7.39.3.60 palDisableLineEvent	448
7.39.3.61 palSetPadCallback	449
7.39.3.62 palSetLineCallback	449
7.39.3.63 PAL_IOPORTS_WIDTH	450
7.39.3.64 PAL_WHOLE_PORT	450
7.39.3.65 PAL_LINE	450
7.39.3.66 PAL_PORT	450
7.39.3.67 PAL_PAD	450
7.39.3.68 PAL_NOLINE	450
7.39.3.69 IOPORT1	450
7.39.3.70 pal_ild_init	450
7.39.3.71 pal_ild_readport	450
7.39.3.72 pal_ild_readlatch	451
7.39.3.73 pal_ild_writeport	451
7.39.3.74 pal_ild_setport	451
7.39.3.75 pal_ild_clearport	452
7.39.3.76 pal_ild_toggleport	452
7.39.3.77 pal_ild_readgroup	453
7.39.3.78 pal_ild_writegroup	453

7.39.3.79	<code>pal_ild_setgroupmode</code>	454
7.39.3.80	<code>pal_ild_readpad</code>	454
7.39.3.81	<code>pal_ild_writepad</code>	455
7.39.3.82	<code>pal_ild_setpad</code>	455
7.39.3.83	<code>pal_ild_clearpad</code>	456
7.39.3.84	<code>pal_ild_togglepad</code>	456
7.39.3.85	<code>pal_ild_setpadmode</code>	457
7.39.3.86	<code>pal_ild_get_pad_event</code>	457
7.39.3.87	<code>pal_ild_get_line_event</code>	457
7.39.4	Typedef Documentation	458
7.39.4.1	<code>palcallback_t</code>	458
7.39.4.2	<code>ioportmask_t</code>	458
7.39.4.3	<code>iomode_t</code>	458
7.39.4.4	<code>ioline_t</code>	458
7.39.4.5	<code>ioportid_t</code>	458
7.39.4.6	<code>iopadid_t</code>	458
7.39.5	Function Documentation	458
7.39.5.1	<code>palReadBus(const IOBus *bus)</code>	458
7.39.5.2	<code>palWriteBus(const IOBus *bus, ioportmask_t bits)</code>	459
7.39.5.3	<code>palSetBusMode(const IOBus *bus, iomode_t mode)</code>	459
7.39.5.4	<code>palSetPadCallbackI(ioportid_t port, iopadid_t pad, palcallback_t cb, void *arg)</code>	460
7.39.5.5	<code>palSetLineCallbackI(ioline_t line, palcallback_t cb, void *arg)</code>	460
7.39.5.6	<code>palWaitPadTimeoutS(ioportid_t port, iopadid_t pad, sysinterval_t timeout)</code>	460
7.39.5.7	<code>palWaitPadTimeout(ioportid_t port, iopadid_t pad, sysinterval_t timeout)</code>	461
7.39.5.8	<code>palWaitLineTimeoutS(ioline_t line, sysinterval_t timeout)</code>	462
7.39.5.9	<code>palWaitLineTimeout(ioline_t line, sysinterval_t timeout)</code>	462
7.39.5.10	<code>_pal_ild_init(const PALConfig *config)</code>	463
7.39.5.11	<code>_pal_ild_setgroupmode(ioportid_t port, ioportmask_t mask, iomode_t mode)</code>	463
7.40	PWM Driver	466
7.40.1	Detailed Description	466
7.40.2	Driver State Machine	466
7.40.3	PWM Operations.	466
7.40.4	Macro Definition Documentation	469
7.40.4.1	<code>PWM_OUTPUT_MASK</code>	469
7.40.4.2	<code>PWM_OUTPUT_DISABLED</code>	469
7.40.4.3	<code>PWM_OUTPUT_ACTIVE_HIGH</code>	469
7.40.4.4	<code>PWM_OUTPUT_ACTIVE_LOW</code>	469
7.40.4.5	<code>PWM_FRACTION_TO_WIDTH</code>	470
7.40.4.6	<code>PWM_DEGREES_TO_WIDTH</code>	470
7.40.4.7	<code>PWM_PERCENTAGE_TO_WIDTH</code>	471

7.40.4.8	pwmChangePeriodl	471
7.40.4.9	pwmEnableChannell	472
7.40.4.10	pwmDisableChannell	472
7.40.4.11	pwmlsChannelEnabledl	473
7.40.4.12	pwmEnablePeriodicNotificationl	473
7.40.4.13	pwmDisablePeriodicNotificationl	474
7.40.4.14	pwmEnableChannelNotificationl	474
7.40.4.15	pwmDisableChannelNotificationl	475
7.40.4.16	PWM_CHANNELS	475
7.40.4.17	PLATFORM_PWM_USE_PWM1	475
7.40.4.18	pwm_llid_change_period	475
7.40.5	Typedef Documentation	476
7.40.5.1	PWMDriver	476
7.40.5.2	pwmcallback_t	476
7.40.5.3	pwmmode_t	476
7.40.5.4	pwmchannel_t	476
7.40.5.5	pwmchnmsk_t	476
7.40.5.6	pwmcnt_t	476
7.40.6	Enumeration Type Documentation	477
7.40.6.1	pwmstate_t	477
7.40.7	Function Documentation	477
7.40.7.1	pwmInit(void)	477
7.40.7.2	pwmObjectInit(PWMDriver *pwmp)	477
7.40.7.3	pwmStart(PWMDriver *pwmp, const PWMConfig *config)	477
7.40.7.4	pwmStop(PWMDriver *pwmp)	478
7.40.7.5	pwmChangePeriod(PWMDriver *pwmp, pwmcnt_t period)	479
7.40.7.6	pwmEnableChannel(PWMDriver *pwmp, pwmchannel_t channel, pwmcnt_t width)	480
7.40.7.7	pwmDisableChannel(PWMDriver *pwmp, pwmchannel_t channel)	481
7.40.7.8	pwmEnablePeriodicNotification(PWMDriver *pwmp)	482
7.40.7.9	pwmDisablePeriodicNotification(PWMDriver *pwmp)	482
7.40.7.10	pwmEnableChannelNotification(PWMDriver *pwmp, pwmchannel_t channel)	483
7.40.7.11	pwmDisableChannelNotification(PWMDriver *pwmp, pwmchannel_t channel)	484
7.40.7.12	pwm_llid_init(void)	484
7.40.7.13	pwm_llid_start(PWMDriver *pwmp)	485
7.40.7.14	pwm_llid_stop(PWMDriver *pwmp)	485
7.40.7.15	pwm_llid_enable_channel(PWMDriver *pwmp, pwmchannel_t channel, pwmcnt_t width)	485
7.40.7.16	pwm_llid_disable_channel(PWMDriver *pwmp, pwmchannel_t channel)	486
7.40.7.17	pwm_llid_enable_periodic_notification(PWMDriver *pwmp)	486
7.40.7.18	pwm_llid_disable_periodic_notification(PWMDriver *pwmp)	487

7.40.7.19	pwm_llc_enable_channel_notification(PWMDriver *pwmp, pwmchannel_t channel)	487
7.40.7.20	pwm_llc_disable_channel_notification(PWMDriver *pwmp, pwmchannel_t channel)	487
7.40.8	Variable Documentation	488
7.40.8.1	PWMD1	488
7.41	QSPI Driver	489
7.41.1	Detailed Description	489
7.41.2	Macro Definition Documentation	492
7.41.2.1	QSPI_USE_WAIT	492
7.41.2.2	QSPI_USE_MUTUAL_EXCLUSION	492
7.41.2.3	qspiStartCommandI	492
7.41.2.4	qspiStartSendI	492
7.41.2.5	qspiStartReceiveI	493
7.41.2.6	qspiMapFlashI	494
7.41.2.7	qspiUnmapFlashI	494
7.41.2.8	_qspi_wakeup_isr	494
7.41.2.9	_qspi_isr_code	495
7.41.2.10	PLATFORM_QSPI_USE_QSPI1	495
7.41.3	Typedef Documentation	495
7.41.3.1	QSPIDriver	495
7.41.3.2	qspicallback_t	496
7.41.4	Enumeration Type Documentation	496
7.41.4.1	qspistate_t	496
7.41.5	Function Documentation	496
7.41.5.1	qspiInit(void)	496
7.41.5.2	qspiObjectInit(QSPIDriver *qspip)	496
7.41.5.3	qspiStart(QSPIDriver *qspip, const QSPIConfig *config)	497
7.41.5.4	qspiStop(QSPIDriver *qspip)	498
7.41.5.5	qspiStartCommand(QSPIDriver *qspip, const qspi_command_t *cmdp)	498
7.41.5.6	qspiStartSend(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)	499
7.41.5.7	qspiStartReceive(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)	499
7.41.5.8	qspiCommand(QSPIDriver *qspip, const qspi_command_t *cmdp)	500
7.41.5.9	qspiSend(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)	501
7.41.5.10	qspiReceive(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)	502
7.41.5.11	qspiMapFlash(QSPIDriver *qspip, const qspi_command_t *cmdp, uint8_t **addrp)	502
7.41.5.12	qspiUnmapFlash(QSPIDriver *qspip)	503
7.41.5.13	qspiAcquireBus(QSPIDriver *qspip)	503
7.41.5.14	qspiReleaseBus(QSPIDriver *qspip)	504

7.41.5.15	qspi_llc_init(void)	505
7.41.5.16	qspi_llc_start(QSPIDriver *qspip)	505
7.41.5.17	qspi_llc_stop(QSPIDriver *qspip)	505
7.41.5.18	qspi_llc_command(QSPIDriver *qspip, const qspi_command_t *cmdp)	505
7.41.5.19	qspi_llc_send(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)	506
7.41.5.20	qspi_llc_receive(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)	506
7.41.5.21	qspi_llc_map_flash(QSPIDriver *qspip, const qspi_command_t *cmdp, uint8_t **addrp)	506
7.41.5.22	qspi_llc_unmap_flash(QSPIDriver *qspip)	507
7.41.6	Variable Documentation	507
7.41.6.1	QSPID1	507
7.42	RTC Driver	508
7.42.1	Detailed Description	508
7.42.2	Macro Definition Documentation	510
7.42.2.1	RTC_BASE_YEAR	510
7.42.2.2	RTC_SUPPORTS_CALLBACKS	510
7.42.2.3	RTC_ALARMS	510
7.42.2.4	RTC_HAS_STORAGE	510
7.42.2.5	PLATFORM_RTC_USE_RTC1	510
7.42.2.6	_rtc_driver_methods	510
7.42.3	Typedef Documentation	510
7.42.3.1	RTCDriver	510
7.42.3.2	rtcalarm_t	510
7.42.3.3	rtccb_t	511
7.42.4	Enumeration Type Documentation	511
7.42.4.1	rtcevent_t	511
7.42.5	Function Documentation	511
7.42.5.1	rtcInit(void)	511
7.42.5.2	rtcObjectInit(RTCDriver *rtcp)	511
7.42.5.3	rtcSetTime(RTCDriver *rtcp, const RTCDateTime *timespec)	511
7.42.5.4	rtcGetTime(RTCDriver *rtcp, RTCDateTime *timespec)	512
7.42.5.5	rtcSetAlarm(RTCDriver *rtcp, rtcalarm_t alarm, const RTCArm *alarmSpec)	513
7.42.5.6	rtcGetAlarm(RTCDriver *rtcp, rtcalarm_t alarm, RTCArm *alarmSpec)	513
7.42.5.7	rtcSetCallback(RTCDriver *rtcp, rtccb_t callback)	514
7.42.5.8	rtcConvertDateTimeToStructTm(const RTCDateTime *timespec, struct tm *timp, uint32_t *tv_msec)	514
7.42.5.9	rtcConvertStructTmToDateTime(const struct tm *timp, uint32_t tv_msec, RTCDateTime *timespec)	515
7.42.5.10	rtcConvertDateTimeToFAT(const RTCDateTime *timespec)	515

7.42.5.11	<code>rtc_lld_init(void)</code>	515
7.42.5.12	<code>rtc_lld_set_time(RTCDriver *rtcp, const RTCDateTime *timespec)</code>	516
7.42.5.13	<code>rtc_lld_get_time(RTCDriver *rtcp, RTCDateTime *timespec)</code>	516
7.42.5.14	<code>rtc_lld_set_alarm(RTCDriver *rtcp, rtcalarm_t alarm, const RTCAalarm *alarmspec)</code>	516
7.42.5.15	<code>rtc_lld_get_alarm(RTCDriver *rtcp, rtcalarm_t alarm, RTCAalarm *alarmspec)</code>	517
7.43	SDC Driver	518
7.43.1	Detailed Description	518
7.43.2	Driver State Machine	518
7.43.3	Driver Operations	518
7.43.4	Macro Definition Documentation	521
7.43.4.1	<code>SDC_INIT_RETRY</code>	521
7.43.4.2	<code>SDC_MMC_SUPPORT</code>	521
7.43.4.3	<code>SDC_NICE_WAITING</code>	522
7.43.4.4	<code>SDC_INIT_OCR_V20</code>	522
7.43.4.5	<code>SDC_INIT_OCR</code>	522
7.43.4.6	<code>sdclsCardInserted</code>	522
7.43.4.7	<code>sdclsWriteProtected</code>	522
7.43.4.8	<code>PLATFORM_SDC_USE_SDC1</code>	523
7.43.4.9	<code>_sd_driver_methods</code>	523
7.43.5	Typedef Documentation	523
7.43.5.1	<code>sdcmode_t</code>	523
7.43.5.2	<code>sdclflags_t</code>	523
7.43.5.3	<code>SDCDriver</code>	523
7.43.6	Enumeration Type Documentation	523
7.43.6.1	<code>mmc_switch_t</code>	523
7.43.6.2	<code>sd_switch_t</code>	524
7.43.6.3	<code>sd_switch_function_t</code>	524
7.43.6.4	<code>sdcbusmode_t</code>	524
7.43.6.5	<code>sdcbusclk_t</code>	524
7.43.7	Function Documentation	524
7.43.7.1	<code>mode_detect(SDCDriver *sdcp)</code>	524
7.43.7.2	<code>mmc_init(SDCDriver *sdcp)</code>	525
7.43.7.3	<code>sd_init(SDCDriver *sdcp)</code>	525
7.43.7.4	<code>mmc_cmd6_construct(mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)</code>	526
7.43.7.5	<code>sd_cmd6_construct(sd_switch_t mode, sd_switch_function_t function, uint32_t value)</code>	526
7.43.7.6	<code>sd_cmd6_extract_info(sd_switch_function_t function, const uint8_t *buf)</code>	527
7.43.7.7	<code>sd_cmd6_check_status(sd_switch_function_t function, const uint8_t *buf)</code>	527
7.43.7.8	<code>sd_detect_bus_clk(SDCDriver *sdcp, sdcbusclk_t *clk)</code>	527

7.43.7.9	<code>mmc_detect_bus_clk(SDCDriver *sdcp, sdcbusclk_t *clk)</code>	528
7.43.7.10	<code>detect_bus_clk(SDCDriver *sdcp, sdcbusclk_t *clk)</code>	529
7.43.7.11	<code>sdcp_set_bus_width(SDCDriver *sdcp)</code>	530
7.43.7.12	<code>mmc_set_bus_width(SDCDriver *sdcp)</code>	531
7.43.7.13	<code>_sdcp_wait_for_transfer_state(SDCDriver *sdcp)</code>	532
7.43.7.14	<code>sdcpInit(void)</code>	532
7.43.7.15	<code>sdcpObjectInit(SDCDriver *sdcp)</code>	533
7.43.7.16	<code>sdcpStart(SDCDriver *sdcp, const SDCCConfig *config)</code>	533
7.43.7.17	<code>sdcpStop(SDCDriver *sdcp)</code>	533
7.43.7.18	<code>sdcpConnect(SDCDriver *sdcp)</code>	534
7.43.7.19	<code>sdcpDisconnect(SDCDriver *sdcp)</code>	535
7.43.7.20	<code>sdcpRead(SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)</code>	536
7.43.7.21	<code>sdcpWrite(SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)</code>	537
7.43.7.22	<code>sdcpGetAndClearErrors(SDCDriver *sdcp)</code>	538
7.43.7.23	<code>sdcpSync(SDCDriver *sdcp)</code>	538
7.43.7.24	<code>sdcpGetInfo(SDCDriver *sdcp, BlockDeviceInfo *bdip)</code>	539
7.43.7.25	<code>sdcpErase(SDCDriver *sdcp, uint32_t startblk, uint32_t endblk)</code>	539
7.43.7.26	<code>sdcp_llid_init(void)</code>	540
7.43.7.27	<code>sdcp_llid_start(SDCDriver *sdcp)</code>	540
7.43.7.28	<code>sdcp_llid_stop(SDCDriver *sdcp)</code>	541
7.43.7.29	<code>sdcp_llid_start_clk(SDCDriver *sdcp)</code>	541
7.43.7.30	<code>sdcp_llid_set_data_clk(SDCDriver *sdcp, sdcbusclk_t clk)</code>	541
7.43.7.31	<code>sdcp_llid_stop_clk(SDCDriver *sdcp)</code>	541
7.43.7.32	<code>sdcp_llid_set_bus_mode(SDCDriver *sdcp, sdcbusmode_t mode)</code>	542
7.43.7.33	<code>sdcp_llid_send_cmd_none(SDCDriver *sdcp, uint8_t cmd, uint32_t arg)</code>	542
7.43.7.34	<code>sdcp_llid_send_cmd_short(SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)</code>	542
7.43.7.35	<code>sdcp_llid_send_cmd_short_crc(SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)</code>	543
7.43.7.36	<code>sdcp_llid_send_cmd_long_crc(SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)</code>	543
7.43.7.37	<code>sdcp_llid_read(SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)</code>	544
7.43.7.38	<code>sdcp_llid_write(SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)</code>	544
7.43.7.39	<code>sdcp_llid_sync(SDCDriver *sdcp)</code>	545
7.43.8	Variable Documentation	545
7.43.8.1	<code>sdcp_vmt</code>	545
7.43.8.2	<code>SDCD1</code>	545
7.44	Generic Sensor Interface	546
7.44.1	Detailed Description	546
7.44.2	Macro Definition Documentation	546
7.44.2.1	<code>_base_sensor_methods_alone</code>	546

7.44.2.2	_base_sensor_methods	547
7.44.2.3	_base_sensor_data	547
7.44.2.4	sensorGetChannelNumber	547
7.44.2.5	sensorReadRaw	547
7.44.2.6	sensorReadCooked	548
7.45	Serial Driver	549
7.45.1	Detailed Description	549
7.45.2	Driver State Machine	549
7.45.3	Macro Definition Documentation	552
7.45.3.1	SD_PARITY_ERROR	552
7.45.3.2	SD_FRAMING_ERROR	552
7.45.3.3	SD_OVERRUN_ERROR	552
7.45.3.4	SD_NOISE_ERROR	552
7.45.3.5	SD_BREAK_DETECTED	552
7.45.3.6	SD_QUEUE_FULL_ERROR	552
7.45.3.7	SERIAL_DEFAULT_BITRATE	552
7.45.3.8	SERIAL_BUFFERS_SIZE	552
7.45.3.9	_serial_driver_methods	553
7.45.3.10	sdPutl	553
7.45.3.11	sdPut	553
7.45.3.12	sdPutTimeout	553
7.45.3.13	sdGetl	553
7.45.3.14	sdGet	554
7.45.3.15	sdGetTimeout	554
7.45.3.16	sdWritel	554
7.45.3.17	sdWrite	554
7.45.3.18	sdWriteTimeout	555
7.45.3.19	sdAsynchronousWrite	555
7.45.3.20	sdReadl	555
7.45.3.21	sdRead	555
7.45.3.22	sdReadTimeout	556
7.45.3.23	sdAsynchronousRead	556
7.45.3.24	PLATFORM_SERIAL_USE_USART1	556
7.45.3.25	_serial_driver_data	556
7.45.4	Typedef Documentation	556
7.45.4.1	SerialDriver	556
7.45.5	Enumeration Type Documentation	557
7.45.5.1	sdstate_t	557
7.45.6	Function Documentation	557
7.45.6.1	sdInit(void)	557

7.45.6.2	<code>sdObjectInit(SerialDriver *sdp, qnotify_t inotify, qnotify_t onotify)</code>	557
7.45.6.3	<code>sdStart(SerialDriver *sdp, const SerialConfig *config)</code>	558
7.45.6.4	<code>sdStop(SerialDriver *sdp)</code>	558
7.45.6.5	<code>sdIncomingData(SerialDriver *sdp, uint8_t b)</code>	559
7.45.6.6	<code>sdRequestData(SerialDriver *sdp)</code>	560
7.45.6.7	<code>sdPutWouldBlock(SerialDriver *sdp)</code>	561
7.45.6.8	<code>sdGetWouldBlock(SerialDriver *sdp)</code>	561
7.45.6.9	<code>sdControl(SerialDriver *sdp, unsigned int operation, void *arg)</code>	562
7.45.6.10	<code>sd_llc_init(void)</code>	563
7.45.6.11	<code>sd_llc_start(SerialDriver *sdp, const SerialConfig *config)</code>	563
7.45.6.12	<code>sd_llc_stop(SerialDriver *sdp)</code>	563
7.45.7	Variable Documentation	564
7.45.7.1	SD1	564
7.45.7.2	default_config	564
7.46	Serial over USB Driver	565
7.46.1	Detailed Description	565
7.46.2	Driver State Machine	565
7.46.3	Macro Definition Documentation	567
7.46.3.1	SERIAL_USB_BUFFERS_SIZE	567
7.46.3.2	SERIAL_USB_BUFFERS_NUMBER	567
7.46.3.3	_serial_usb_driver_data	567
7.46.3.4	_serial_usb_driver_methods	567
7.46.4	Typedef Documentation	567
7.46.4.1	SerialUSBDriver	567
7.46.5	Enumeration Type Documentation	568
7.46.5.1	sdustate_t	568
7.46.6	Function Documentation	568
7.46.6.1	ibnotify(io_buffers_queue_t *bqp)	568
7.46.6.2	obnotify(io_buffers_queue_t *bqp)	568
7.46.6.3	sduInit(void)	568
7.46.6.4	sduObjectInit(SerialUSBDriver *sdup)	569
7.46.6.5	sduStart(SerialUSBDriver *sdup, const SerialUSBConfig *config)	569
7.46.6.6	sduStop(SerialUSBDriver *sdup)	570
7.46.6.7	sduSuspendHook(SerialUSBDriver *sdup)	571
7.46.6.8	sduWakeupHook(SerialUSBDriver *sdup)	571
7.46.6.9	sduConfigureHook(SerialUSBDriver *sdup)	571
7.46.6.10	sduRequestsHook(USBDriver *usbp)	572
7.46.6.11	sduSOFHook(SerialUSBDriver *sdup)	572
7.46.6.12	sduDataTransmitted(USBDriver *usbp, usbp_t ep)	573
7.46.6.13	sduDataReceived(USBDriver *usbp, usbp_t ep)	573

7.46.6.14	sduInterruptTransmitted(USBDriver *usbp, usbep_t ep)	574
7.46.6.15	sduControl(USBDriver *usbp, unsigned int operation, void *arg)	574
7.47	SPI Driver	576
7.47.1	Detailed Description	576
7.47.2	Driver State Machine	576
7.47.3	Macro Definition Documentation	579
7.47.3.1	SPI_USE_WAIT	579
7.47.3.2	SPI_USE_CIRCULAR	579
7.47.3.3	SPI_USE_MUTUAL_EXCLUSION	580
7.47.3.4	SPI_SELECT_MODE	580
7.47.3.5	spiSelectI	580
7.47.3.6	spiUnselectI	580
7.47.3.7	spiStartIgnoreI	581
7.47.3.8	spiStartExchangeI	581
7.47.3.9	spiStartSendI	582
7.47.3.10	spiStartReceiveI	582
7.47.3.11	spiPolledExchange	583
7.47.3.12	_spi_wakeup_isr	584
7.47.3.13	_spi_isr_code	584
7.47.3.14	_spi_isr_code_half1	585
7.47.3.15	_spi_isr_code_half2	585
7.47.3.16	SPI_SUPPORTS_CIRCULAR	586
7.47.3.17	PLATFORM_SPI_USE_SPI1	586
7.47.4	Typedef Documentation	586
7.47.4.1	SPIDriver	586
7.47.4.2	spicallback_t	586
7.47.5	Enumeration Type Documentation	586
7.47.5.1	spistate_t	586
7.47.6	Function Documentation	586
7.47.6.1	spiInit(void)	586
7.47.6.2	spiObjectInit(SPIDriver *spip)	587
7.47.6.3	spiStart(SPIDriver *spip, const SPICConfig *config)	587
7.47.6.4	spiStop(SPIDriver *spip)	588
7.47.6.5	spiSelect(SPIDriver *spip)	589
7.47.6.6	spiUnselect(SPIDriver *spip)	589
7.47.6.7	spiStartIgnore(SPIDriver *spip, size_t n)	590
7.47.6.8	spiStartExchange(SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)	590
7.47.6.9	spiStartSend(SPIDriver *spip, size_t n, const void *txbuf)	591
7.47.6.10	spiStartReceive(SPIDriver *spip, size_t n, void *rxbuf)	592
7.47.6.11	spiAbortI(SPIDriver *spip)	593

7.47.6.12	<code>spiAbort(SPIDriver *spip)</code>	593
7.47.6.13	<code>spiIgnore(SPIDriver *spip, size_t n)</code>	594
7.47.6.14	<code>spiExchange(SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)</code>	595
7.47.6.15	<code>spiSend(SPIDriver *spip, size_t n, const void *txbuf)</code>	596
7.47.6.16	<code>spiReceive(SPIDriver *spip, size_t n, void *rxbuf)</code>	597
7.47.6.17	<code>spiAcquireBus(SPIDriver *spip)</code>	598
7.47.6.18	<code>spiReleaseBus(SPIDriver *spip)</code>	598
7.47.6.19	<code>spi_ll_init(void)</code>	599
7.47.6.20	<code>spi_ll_start(SPIDriver *spip)</code>	599
7.47.6.21	<code>spi_ll_stop(SPIDriver *spip)</code>	600
7.47.6.22	<code>spi_ll_select(SPIDriver *spip)</code>	600
7.47.6.23	<code>spi_ll_deselect(SPIDriver *spip)</code>	600
7.47.6.24	<code>spi_ll_ignore(SPIDriver *spip, size_t n)</code>	600
7.47.6.25	<code>spi_ll_exchange(SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)</code>	601
7.47.6.26	<code>spi_ll_send(SPIDriver *spip, size_t n, const void *txbuf)</code>	601
7.47.6.27	<code>spi_ll_receive(SPIDriver *spip, size_t n, void *rxbuf)</code>	602
7.47.6.28	<code>spi_ll_abort(SPIDriver *spip)</code>	602
7.47.6.29	<code>spi_ll_polled_exchange(SPIDriver *spip, uint16_t frame)</code>	602
7.47.7	Variable Documentation	603
7.47.7.1	SPID1	603
7.48	ST Driver	604
7.48.1	Detailed Description	604
7.48.2	Macro Definition Documentation	604
7.48.2.1	<code>stGetCounter</code>	604
7.48.2.2	<code>stIsAlarmActive</code>	605
7.48.3	Function Documentation	605
7.48.3.1	<code>stInit(void)</code>	605
7.48.3.2	<code>stStartAlarm(sys_time_t abstime)</code>	606
7.48.3.3	<code>stStopAlarm(void)</code>	606
7.48.3.4	<code>stSetAlarm(sys_time_t abstime)</code>	607
7.48.3.5	<code>stGetAlarm(void)</code>	607
7.48.3.6	<code>st_ll_init(void)</code>	608
7.48.3.7	<code>st_ll_get_counter(void)</code>	608
7.48.3.8	<code>st_ll_start_alarm(sys_time_t abstime)</code>	608
7.48.3.9	<code>st_ll_stop_alarm(void)</code>	608
7.48.3.10	<code>st_ll_set_alarm(sys_time_t abstime)</code>	608
7.48.3.11	<code>st_ll_get_alarm(void)</code>	609
7.48.3.12	<code>st_ll_is_alarm_active(void)</code>	609
7.49	Generic Thermometer Interface	610
7.49.1	Detailed Description	610

7.49.2	Macro Definition Documentation	610
7.49.2.1	<code>_base_thermometer_methods_alone</code>	610
7.49.2.2	<code>_base_thermometer_methods</code>	611
7.49.2.3	<code>_base_thermometer_data</code>	611
7.49.2.4	<code>thermometerGetChannelsNumber</code>	611
7.49.2.5	<code>thermometerReadRaw</code>	611
7.49.2.6	<code>thermometerReadCooked</code>	612
7.49.2.7	<code>thermometerSetBias</code>	612
7.49.2.8	<code>thermometerResetBias</code>	613
7.49.2.9	<code>thermometerSetSensitivity</code>	613
7.49.2.10	<code>thermometerResetSensitivity</code>	614
7.50	UART Driver	615
7.50.1	Detailed Description	615
7.50.2	Driver State Machine	615
7.50.2.1	Transmitter sub State Machine	616
7.50.2.2	Receiver sub State Machine	616
7.50.3	Macro Definition Documentation	619
7.50.3.1	<code>UART_NO_ERROR</code>	619
7.50.3.2	<code>UART_PARITY_ERROR</code>	619
7.50.3.3	<code>UART_FRAMING_ERROR</code>	619
7.50.3.4	<code>UART_OVERRUN_ERROR</code>	619
7.50.3.5	<code>UART_NOISE_ERROR</code>	619
7.50.3.6	<code>UART_BREAK_DETECTED</code>	620
7.50.3.7	<code>UART_USE_WAIT</code>	620
7.50.3.8	<code>UART_USE_MUTUAL_EXCLUSION</code>	620
7.50.3.9	<code>_uart_wakeup_tx1_isr</code>	620
7.50.3.10	<code>_uart_wakeup_tx2_isr</code>	620
7.50.3.11	<code>_uart_wakeup_rx_complete_isr</code>	622
7.50.3.12	<code>_uart_wakeup_rx_error_isr</code>	622
7.50.3.13	<code>_uart_wakeup_rx_timeout_isr</code>	622
7.50.3.14	<code>_uart_tx1_isr_code</code>	623
7.50.3.15	<code>_uart_tx2_isr_code</code>	623
7.50.3.16	<code>_uart_rx_complete_isr_code</code>	624
7.50.3.17	<code>_uart_rx_error_isr_code</code>	625
7.50.3.18	<code>_uart_rx_idle_code</code>	625
7.50.3.19	<code>_uart_timeout_isr_code</code>	626
7.50.3.20	<code>PLATFORM_UART_USE_UART1</code>	626
7.50.4	Typedef Documentation	627
7.50.4.1	<code>uartflags_t</code>	627
7.50.4.2	<code>UARTDriver</code>	627

7.50.4.3	uartcb_t	627
7.50.4.4	uartccb_t	627
7.50.4.5	uartecb_t	627
7.50.5	Enumeration Type Documentation	627
7.50.5.1	uartstate_t	627
7.50.5.2	uarttxstate_t	628
7.50.5.3	uartrxstate_t	628
7.50.6	Function Documentation	628
7.50.6.1	uartInit(void)	628
7.50.6.2	uartObjectInit(UARTDriver *uartp)	628
7.50.6.3	uartStart(UARTDriver *uartp, const UARTConfig *config)	629
7.50.6.4	uartStop(UARTDriver *uartp)	630
7.50.6.5	uartStartSend(UARTDriver *uartp, size_t n, const void *txbuf)	630
7.50.6.6	uartStartSendI(UARTDriver *uartp, size_t n, const void *txbuf)	631
7.50.6.7	uartStopSend(UARTDriver *uartp)	631
7.50.6.8	uartStopSendI(UARTDriver *uartp)	632
7.50.6.9	uartStartReceive(UARTDriver *uartp, size_t n, void *rxbuf)	633
7.50.6.10	uartStartReceiveI(UARTDriver *uartp, size_t n, void *rxbuf)	634
7.50.6.11	uartStopReceive(UARTDriver *uartp)	634
7.50.6.12	uartStopReceiveI(UARTDriver *uartp)	635
7.50.6.13	uartSendTimeout(UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval↔ _t timeout)	636
7.50.6.14	uartSendFullTimeout(UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)	637
7.50.6.15	uartReceiveTimeout(UARTDriver *uartp, size_t *np, void *rxbuf, sysinterval↔ t timeout)	638
7.50.6.16	uartAcquireBus(UARTDriver *uartp)	639
7.50.6.17	uartReleaseBus(UARTDriver *uartp)	639
7.50.6.18	uart_lld_init(void)	640
7.50.6.19	uart_lld_start(UARTDriver *uartp)	640
7.50.6.20	uart_lld_stop(UARTDriver *uartp)	641
7.50.6.21	uart_lld_start_send(UARTDriver *uartp, size_t n, const void *txbuf)	641
7.50.6.22	uart_lld_stop_send(UARTDriver *uartp)	641
7.50.6.23	uart_lld_start_receive(UARTDriver *uartp, size_t n, void *rxbuf)	642
7.50.6.24	uart_lld_stop_receive(UARTDriver *uartp)	642
7.50.7	Variable Documentation	642
7.50.7.1	UARTD1	642
7.51	USB Driver	643
7.51.1	Detailed Description	643
7.51.2	Driver State Machine	643
7.51.3	USB Operations	643

7.51.3.1	USB Implementation	643
7.51.3.2	USB Endpoints	644
7.51.3.3	USB Callbacks	645
7.51.4	Macro Definition Documentation	649
7.51.4.1	USB_DESC_INDEX	649
7.51.4.2	USB_DESC_BYTE	650
7.51.4.3	USB_DESC_WORD	650
7.51.4.4	USB_DESC_BCD	650
7.51.4.5	USB_DESC_DEVICE	650
7.51.4.6	USB_DESC_CONFIGURATION_SIZE	650
7.51.4.7	USB_DESC_CONFIGURATION	651
7.51.4.8	USB_DESC_INTERFACE_SIZE	651
7.51.4.9	USB_DESC_INTERFACE	651
7.51.4.10	USB_DESC_INTERFACE_ASSOCIATION_SIZE	651
7.51.4.11	USB_DESC_INTERFACE_ASSOCIATION	651
7.51.4.12	USB_DESC_ENDPOINT_SIZE	652
7.51.4.13	USB_DESC_ENDPOINT	652
7.51.4.14	USB_EP_MODE_TYPE	652
7.51.4.15	USB_EP_MODE_TYPE_CTRL	652
7.51.4.16	USB_EP_MODE_TYPE_ISOC	652
7.51.4.17	USB_EP_MODE_TYPE_BULK	652
7.51.4.18	USB_EP_MODE_TYPE_INTR	652
7.51.4.19	USB_USE_WAIT	653
7.51.4.20	usbGetDriverStatel	653
7.51.4.21	usbConnectBus	653
7.51.4.22	usbDisconnectBus	653
7.51.4.23	usbGetFrameNumberX	653
7.51.4.24	usbGetTransmitStatusl	654
7.51.4.25	usbGetReceiveStatusl	654
7.51.4.26	usbGetReceiveTransactionSizeX	655
7.51.4.27	usbSetupTransfer	655
7.51.4.28	usbReadSetup	656
7.51.4.29	_usb_isr_invoke_event_cb	656
7.51.4.30	_usb_isr_invoke_sof_cb	656
7.51.4.31	_usb_isr_invoke_setup_cb	657
7.51.4.32	_usb_isr_invoke_in_cb	657
7.51.4.33	_usb_isr_invoke_out_cb	658
7.51.4.34	USB_MAX_ENDPOINTS	658
7.51.4.35	USB_EP0_STATUS_STAGE	658
7.51.4.36	USB_SET_ADDRESS_MODE	658

7.51.4.37	USB_SET_ADDRESS_ACK_HANDLING	658
7.51.4.38	PLATFORM_USB_USE_USB1	658
7.51.4.39	usb_lld_get_frame_number	658
7.51.4.40	usb_lld_get_transaction_size	659
7.51.4.41	usb_lld_connect_bus	659
7.51.4.42	usb_lld_disconnect_bus	659
7.51.4.43	usb_lld_wakeup_host	660
7.51.5	Typedef Documentation	660
7.51.5.1	USBDriver	660
7.51.5.2	usbep_t	660
7.51.5.3	usbcallback_t	660
7.51.5.4	usbepcallback_t	660
7.51.5.5	usbeventcb_t	660
7.51.5.6	usbreqhandler_t	660
7.51.5.7	usbgetdescriptor_t	661
7.51.6	Enumeration Type Documentation	661
7.51.6.1	usbstate_t	661
7.51.6.2	usbepstatus_t	661
7.51.6.3	usbep0state_t	661
7.51.6.4	usbevent_t	662
7.51.7	Function Documentation	662
7.51.7.1	set_address(USBDriver *usbp)	662
7.51.7.2	default_handler(USBDriver *usbp)	662
7.51.7.3	usbInit(void)	663
7.51.7.4	usbObjectInit(USBDriver *usbp)	664
7.51.7.5	usbStart(USBDriver *usbp, const USBConfig *config)	664
7.51.7.6	usbStop(USBDriver *usbp)	665
7.51.7.7	usbInitEndpointI(USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)	666
7.51.7.8	usbDisableEndpointI(USBDriver *usbp)	666
7.51.7.9	usbStartReceiveI(USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)	667
7.51.7.10	usbStartTransmitI(USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)	667
7.51.7.11	usbReceive(USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)	668
7.51.7.12	usbTransmit(USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)	669
7.51.7.13	usbStallReceiveI(USBDriver *usbp, usbep_t ep)	670
7.51.7.14	usbStallTransmitI(USBDriver *usbp, usbep_t ep)	671
7.51.7.15	usbWakeupHost(USBDriver *usbp)	671
7.51.7.16	_usb_reset(USBDriver *usbp)	672
7.51.7.17	_usb_suspend(USBDriver *usbp)	672
7.51.7.18	_usb_wakeup(USBDriver *usbp)	673
7.51.7.19	_usb_ep0setup(USBDriver *usbp, usbep_t ep)	673

7.51.7.20	_usb_ep0in(USBDriver *usbp, usbep_t ep)	674
7.51.7.21	_usb_ep0out(USBDriver *usbp, usbep_t ep)	675
7.51.7.22	usb_llid_init(void)	676
7.51.7.23	usb_llid_start(USBDriver *usbp)	676
7.51.7.24	usb_llid_stop(USBDriver *usbp)	677
7.51.7.25	usb_llid_reset(USBDriver *usbp)	677
7.51.7.26	usb_llid_set_address(USBDriver *usbp)	677
7.51.7.27	usb_llid_init_endpoint(USBDriver *usbp, usbep_t ep)	677
7.51.7.28	usb_llid_disable_endpoints(USBDriver *usbp)	678
7.51.7.29	usb_llid_get_status_out(USBDriver *usbp, usbep_t ep)	678
7.51.7.30	usb_llid_get_status_in(USBDriver *usbp, usbep_t ep)	678
7.51.7.31	usb_llid_read_setup(USBDriver *usbp, usbep_t ep, uint8_t *buf)	679
7.51.7.32	usb_llid_prepare_receive(USBDriver *usbp, usbep_t ep)	679
7.51.7.33	usb_llid_prepare_transmit(USBDriver *usbp, usbep_t ep)	680
7.51.7.34	usb_llid_start_out(USBDriver *usbp, usbep_t ep)	680
7.51.7.35	usb_llid_start_in(USBDriver *usbp, usbep_t ep)	680
7.51.7.36	usb_llid_stall_out(USBDriver *usbp, usbep_t ep)	680
7.51.7.37	usb_llid_stall_in(USBDriver *usbp, usbep_t ep)	681
7.51.7.38	usb_llid_clear_out(USBDriver *usbp, usbep_t ep)	681
7.51.7.39	usb_llid_clear_in(USBDriver *usbp, usbep_t ep)	681
7.51.8	Variable Documentation	681
7.51.8.1	USBID1	681
7.51.8.2	ep0_state	682
7.51.8.3	in	682
7.51.8.4	out	682
7.51.8.5	ep0config	682
7.52	USB CDC Header	683
7.52.1	Detailed Description	683
7.53	WDG Driver	685
7.53.1	Detailed Description	685
7.53.2	Macro Definition Documentation	686
7.53.2.1	wdgResetI	686
7.53.2.2	PLATFORM_WDG_USE_WDG1	686
7.53.3	Typedef Documentation	686
7.53.3.1	WDGDriver	686
7.53.4	Enumeration Type Documentation	686
7.53.4.1	wdgstate_t	686
7.53.5	Function Documentation	686
7.53.5.1	wdgInit(void)	686
7.53.5.2	wdgStart(WDGDriver *wdgp, const WDGConfig *config)	687

7.53.5.3	wdgStop(WDGDriver *wdgp)	687
7.53.5.4	wdgReset(WDGDriver *wdgp)	688
7.53.5.5	wdg_lld_init(void)	689
7.53.5.6	wdg_lld_start(WDGDriver *wdgp)	689
7.53.5.7	wdg_lld_stop(WDGDriver *wdgp)	689
7.53.5.8	wdg_lld_reset(WDGDriver *wdgp)	689
8	Data Structure Documentation	691
8.1	ADCConfig Struct Reference	691
8.1.1	Detailed Description	691
8.2	ADCCConversionGroup Struct Reference	692
8.2.1	Detailed Description	692
8.2.2	Field Documentation	693
8.2.2.1	circular	693
8.2.2.2	num_channels	693
8.2.2.3	end_cb	693
8.2.2.4	error_cb	693
8.3	ADCDriver Struct Reference	693
8.3.1	Detailed Description	694
8.3.2	Field Documentation	695
8.3.2.1	state	695
8.3.2.2	config	695
8.3.2.3	samples	695
8.3.2.4	depth	695
8.3.2.5	grpp	695
8.3.2.6	thread	695
8.3.2.7	mutex	695
8.4	BaseAccelerometer Struct Reference	695
8.4.1	Detailed Description	697
8.4.2	Field Documentation	697
8.4.2.1	vmt	697
8.5	BaseAccelerometerVMT Struct Reference	697
8.5.1	Detailed Description	698
8.6	BaseAsynchronousChannel Struct Reference	698
8.6.1	Detailed Description	700
8.6.2	Field Documentation	700
8.6.2.1	vmt	700
8.7	BaseAsynchronousChannelVMT Struct Reference	700
8.7.1	Detailed Description	702
8.8	BaseBarometer Struct Reference	702

8.8.1	Detailed Description	704
8.8.2	Field Documentation	704
8.8.2.1	vmt	704
8.9	BaseBarometerVMT Struct Reference	704
8.9.1	Detailed Description	705
8.10	BaseBlockDevice Struct Reference	705
8.10.1	Detailed Description	707
8.10.2	Field Documentation	707
8.10.2.1	vmt	707
8.11	BaseBlockDeviceVMT Struct Reference	707
8.11.1	Detailed Description	708
8.12	BaseChannel Struct Reference	708
8.12.1	Detailed Description	710
8.12.2	Field Documentation	710
8.12.2.1	vmt	710
8.13	BaseChannelVMT Struct Reference	710
8.13.1	Detailed Description	712
8.14	BaseCompass Struct Reference	712
8.14.1	Detailed Description	713
8.14.2	Field Documentation	713
8.14.2.1	vmt	713
8.15	BaseCompassVMT Struct Reference	713
8.15.1	Detailed Description	714
8.16	BaseFlash Struct Reference	714
8.16.1	Detailed Description	716
8.16.2	Field Documentation	716
8.16.2.1	vmt	716
8.17	BaseFlashVMT Struct Reference	716
8.17.1	Detailed Description	717
8.18	BaseGyroscope Struct Reference	717
8.18.1	Detailed Description	718
8.18.2	Field Documentation	718
8.18.2.1	vmt	718
8.19	BaseGyroscopeVMT Struct Reference	718
8.19.1	Detailed Description	719
8.20	BaseHygrometer Struct Reference	719
8.20.1	Detailed Description	720
8.20.2	Field Documentation	720
8.20.2.1	vmt	720
8.21	BaseHygrometerVMT Struct Reference	720

8.21.1 Detailed Description	721
8.22 BaseObject Struct Reference	721
8.22.1 Detailed Description	722
8.22.2 Field Documentation	722
8.22.2.1 vmt	722
8.23 BaseObjectVMT Struct Reference	722
8.23.1 Detailed Description	723
8.24 BaseSensor Struct Reference	723
8.24.1 Detailed Description	724
8.24.2 Field Documentation	724
8.24.2.1 vmt	724
8.25 BaseSensorVMT Struct Reference	724
8.25.1 Detailed Description	725
8.26 BaseSequentialStream Struct Reference	725
8.26.1 Detailed Description	727
8.26.2 Field Documentation	727
8.26.2.1 vmt	727
8.27 BaseSequentialStreamVMT Struct Reference	727
8.27.1 Detailed Description	728
8.28 BaseThermometer Struct Reference	728
8.28.1 Detailed Description	730
8.28.2 Field Documentation	730
8.28.2.1 vmt	730
8.29 BaseThermometerVMT Struct Reference	730
8.29.1 Detailed Description	731
8.30 BlockDeviceInfo Struct Reference	731
8.30.1 Detailed Description	731
8.30.2 Field Documentation	732
8.30.2.1 blk_size	732
8.30.2.2 blk_num	732
8.31 CANConfig Struct Reference	732
8.31.1 Detailed Description	732
8.32 CANDriver Struct Reference	732
8.32.1 Detailed Description	733
8.32.2 Field Documentation	734
8.32.2.1 state	734
8.32.2.2 config	734
8.32.2.3 txqueue	734
8.32.2.4 rxqueue	734
8.32.2.5 rxfull_event	734

8.32.2.6	txempty_event	734
8.32.2.7	error_event	734
8.32.2.8	sleep_event	734
8.32.2.9	wakeup_event	734
8.33	CANRxFrame Struct Reference	735
8.33.1	Detailed Description	736
8.33.2	Field Documentation	736
8.33.2.1	FMI	736
8.33.2.2	TIME	736
8.33.2.3	DLC	736
8.33.2.4	RTR	736
8.33.2.5	IDE	736
8.33.2.6	SID	736
8.33.2.7	EID	736
8.33.2.8	data8	736
8.33.2.9	data16	736
8.33.2.10	data32	736
8.34	CANTxFrame Struct Reference	737
8.34.1	Detailed Description	737
8.34.2	Field Documentation	738
8.34.2.1	DLC	738
8.34.2.2	RTR	738
8.34.2.3	IDE	738
8.34.2.4	SID	738
8.34.2.5	EID	738
8.34.2.6	data8	738
8.34.2.7	data16	738
8.34.2.8	data32	738
8.35	cdc_linecoding_t Struct Reference	738
8.35.1	Detailed Description	739
8.36	CRYConfig Struct Reference	739
8.36.1	Detailed Description	739
8.37	CRYDriver Struct Reference	739
8.37.1	Detailed Description	740
8.37.2	Field Documentation	740
8.37.2.1	state	740
8.37.2.2	config	740
8.37.2.3	key0_type	741
8.37.2.4	key0_size	741
8.37.2.5	key0_buffer	741

8.38 DACConfig Struct Reference	741
8.38.1 Detailed Description	741
8.39 DACConversionGroup Struct Reference	741
8.39.1 Detailed Description	742
8.39.2 Field Documentation	742
8.39.2.1 num_channels	742
8.39.2.2 end_cb	742
8.39.2.3 error_cb	742
8.40 DACDriver Struct Reference	743
8.40.1 Detailed Description	744
8.40.2 Field Documentation	744
8.40.2.1 state	744
8.40.2.2 grpp	744
8.40.2.3 samples	744
8.40.2.4 depth	744
8.40.2.5 config	744
8.40.2.6 thread	744
8.40.2.7 mutex	744
8.41 event_source Struct Reference	744
8.41.1 Detailed Description	745
8.41.2 Field Documentation	745
8.41.2.1 flags	745
8.41.2.2 cb	745
8.41.2.3 param	745
8.42 EXTChannelConfig Struct Reference	745
8.42.1 Detailed Description	746
8.42.2 Field Documentation	746
8.42.2.1 mode	746
8.42.2.2 cb	746
8.43 EXTConfig Struct Reference	747
8.43.1 Detailed Description	747
8.43.2 Field Documentation	747
8.43.2.1 channels	747
8.44 EXTDriver Struct Reference	748
8.44.1 Detailed Description	748
8.44.2 Field Documentation	748
8.44.2.1 state	748
8.44.2.2 config	749
8.45 FileStream Struct Reference	749
8.45.1 Detailed Description	750

8.45.2	Field Documentation	750
8.45.2.1	vmt	750
8.46	FileStreamVMT Struct Reference	750
8.46.1	Detailed Description	751
8.47	flash_descriptor_t Struct Reference	751
8.47.1	Detailed Description	752
8.47.2	Field Documentation	752
8.47.2.1	attributes	752
8.47.2.2	page_size	753
8.47.2.3	sectors_count	753
8.47.2.4	sectors	753
8.47.2.5	sectors_size	753
8.47.2.6	address	753
8.48	flash_sector_descriptor_t Struct Reference	753
8.48.1	Detailed Description	754
8.48.2	Field Documentation	754
8.48.2.1	offset	754
8.48.2.2	size	754
8.49	GPTConfig Struct Reference	754
8.49.1	Detailed Description	755
8.49.2	Field Documentation	755
8.49.2.1	frequency	755
8.49.2.2	callback	755
8.50	GPTDriver Struct Reference	755
8.50.1	Detailed Description	756
8.50.2	Field Documentation	756
8.50.2.1	state	756
8.50.2.2	config	756
8.51	I2CConfig Struct Reference	756
8.51.1	Detailed Description	757
8.52	I2CDriver Struct Reference	757
8.52.1	Detailed Description	758
8.52.2	Field Documentation	758
8.52.2.1	state	758
8.52.2.2	config	758
8.52.2.3	errors	758
8.53	I2SConfig Struct Reference	758
8.53.1	Detailed Description	759
8.53.2	Field Documentation	759
8.53.2.1	tx_buffer	759

8.53.2.2	rx_buffer	759
8.53.2.3	size	759
8.53.2.4	end_cb	759
8.54	I2SDriver Struct Reference	759
8.54.1	Detailed Description	760
8.54.2	Field Documentation	760
8.54.2.1	state	760
8.54.2.2	config	760
8.55	ICUConfig Struct Reference	760
8.55.1	Detailed Description	761
8.55.2	Field Documentation	761
8.55.2.1	mode	761
8.55.2.2	frequency	762
8.55.2.3	width_cb	762
8.55.2.4	period_cb	762
8.55.2.5	overflow_cb	762
8.56	ICUDriver Struct Reference	762
8.56.1	Detailed Description	763
8.56.2	Field Documentation	763
8.56.2.1	state	763
8.56.2.2	config	763
8.57	io_buffers_queue Struct Reference	763
8.57.1	Detailed Description	764
8.57.2	Field Documentation	764
8.57.2.1	waiting	764
8.57.2.2	suspended	764
8.57.2.3	bcounter	764
8.57.2.4	bwrptr	764
8.57.2.5	brdptr	765
8.57.2.6	btot	765
8.57.2.7	bsize	765
8.57.2.8	bn	765
8.57.2.9	buffers	765
8.57.2.10	ptr	765
8.57.2.11	top	765
8.57.2.12	notify	765
8.57.2.13	link	765
8.58	io_queue Struct Reference	765
8.58.1	Detailed Description	766
8.58.2	Field Documentation	767

8.58.2.1	q_waiting	767
8.58.2.2	q_counter	767
8.58.2.3	q_buffer	767
8.58.2.4	q_top	767
8.58.2.5	q_wrptr	767
8.58.2.6	q_rdptr	767
8.58.2.7	q_notify	767
8.58.2.8	q_link	767
8.59	IOBus Struct Reference	767
8.59.1	Detailed Description	768
8.59.2	Field Documentation	768
8.59.2.1	portid	768
8.59.2.2	mask	768
8.59.2.3	offset	768
8.60	JESD215Flash Struct Reference	769
8.60.1	Detailed Description	769
8.60.2	Field Documentation	769
8.60.2.1	vmt	769
8.61	JESD215FlashVMT Struct Reference	769
8.61.1	Detailed Description	770
8.62	MACConfig Struct Reference	770
8.62.1	Detailed Description	770
8.62.2	Field Documentation	770
8.62.2.1	mac_address	770
8.63	MACDriver Struct Reference	771
8.63.1	Detailed Description	771
8.63.2	Field Documentation	771
8.63.2.1	state	771
8.63.2.2	config	772
8.63.2.3	tdqueue	772
8.63.2.4	rdqueue	772
8.63.2.5	rdevent	772
8.64	MACReceiveDescriptor Struct Reference	772
8.64.1	Detailed Description	772
8.64.2	Field Documentation	773
8.64.2.1	offset	773
8.64.2.2	size	773
8.65	MACTransmitDescriptor Struct Reference	773
8.65.1	Detailed Description	773
8.65.2	Field Documentation	773

8.65.2.1	offset	773
8.65.2.2	size	773
8.66	mfs_bank_header_t Union Reference	774
8.66.1	Detailed Description	774
8.66.2	Field Documentation	774
8.66.2.1	magic1	774
8.66.2.2	magic2	774
8.66.2.3	counter	774
8.66.2.4	reserved1	774
8.66.2.5	crc	775
8.67	mfs_data_header_t Union Reference	775
8.67.1	Detailed Description	775
8.67.2	Field Documentation	775
8.67.2.1	magic	775
8.67.2.2	id	775
8.67.2.3	crc	775
8.67.2.4	size	775
8.68	MFSCfg Struct Reference	776
8.68.1	Detailed Description	777
8.68.2	Field Documentation	777
8.68.2.1	flashp	777
8.68.2.2	erased	777
8.68.2.3	bank_size	777
8.68.2.4	bank0_start	777
8.68.2.5	bank0_sectors	777
8.68.2.6	bank1_start	777
8.68.2.7	bank1_sectors	777
8.69	MFSDriver Struct Reference	777
8.69.1	Detailed Description	780
8.69.2	Field Documentation	780
8.69.2.1	state	780
8.69.2.2	config	780
8.69.2.3	current_bank	780
8.69.2.4	current_counter	780
8.69.2.5	next_offset	780
8.69.2.6	used_space	780
8.69.2.7	descriptors	780
8.69.2.8	buffer	780
8.70	MMCCfg Struct Reference	781
8.70.1	Detailed Description	781

8.70.2	Field Documentation	782
8.70.2.1	spip	782
8.70.2.2	lscfg	782
8.70.2.3	hscfg	782
8.71	MMCDriver Struct Reference	782
8.71.1	Detailed Description	783
8.71.2	Field Documentation	783
8.71.2.1	vmt	783
8.71.2.2	config	784
8.72	MMCDriverVMT Struct Reference	784
8.72.1	Detailed Description	785
8.73	MMCSDBlockDevice Struct Reference	785
8.73.1	Detailed Description	787
8.73.2	Field Documentation	787
8.73.2.1	vmt	787
8.74	MMCSDBlockDeviceVMT Struct Reference	787
8.74.1	Detailed Description	788
8.75	PALConfig Struct Reference	789
8.75.1	Detailed Description	789
8.76	palevent_t Struct Reference	789
8.76.1	Detailed Description	790
8.76.2	Field Documentation	790
8.76.2.1	threads	790
8.76.2.2	cb	790
8.76.2.3	arg	790
8.77	PWMChannelConfig Struct Reference	791
8.77.1	Detailed Description	791
8.77.2	Field Documentation	791
8.77.2.1	mode	791
8.77.2.2	callback	792
8.78	PWMConfig Struct Reference	792
8.78.1	Detailed Description	793
8.78.2	Field Documentation	793
8.78.2.1	frequency	793
8.78.2.2	period	793
8.78.2.3	callback	793
8.78.2.4	channels	793
8.79	PWMDriver Struct Reference	793
8.79.1	Detailed Description	794
8.79.2	Field Documentation	794

8.79.2.1	state	795
8.79.2.2	config	795
8.79.2.3	period	795
8.79.2.4	enabled	795
8.79.2.5	channels	795
8.80	qspi_command_t Struct Reference	795
8.80.1	Detailed Description	795
8.81	QSPIDriver Struct Reference	795
8.81.1	Detailed Description	796
8.81.2	Field Documentation	796
8.81.2.1	end_cb	796
8.82	QSPIDriver Struct Reference	796
8.82.1	Detailed Description	797
8.82.2	Field Documentation	797
8.82.2.1	state	797
8.82.2.2	config	797
8.82.2.3	thread	797
8.82.2.4	mutex	798
8.83	RTCArm Struct Reference	798
8.83.1	Detailed Description	798
8.84	RTCDateTime Struct Reference	798
8.84.1	Detailed Description	799
8.84.2	Field Documentation	799
8.84.2.1	year	799
8.84.2.2	month	799
8.84.2.3	dstflag	799
8.84.2.4	dayofweek	799
8.84.2.5	day	799
8.84.2.6	millisecond	799
8.85	RTCDriver Struct Reference	800
8.85.1	Detailed Description	800
8.85.2	Field Documentation	800
8.85.2.1	vmt	800
8.86	RTCDriverVMT Struct Reference	800
8.86.1	Detailed Description	802
8.87	SDCConfig Struct Reference	802
8.87.1	Detailed Description	803
8.87.2	Field Documentation	803
8.87.2.1	scratchpad	803
8.87.2.2	bus_width	803

8.88	SDCDriver Struct Reference	803
8.88.1	Detailed Description	804
8.88.2	Field Documentation	804
8.88.2.1	vmt	804
8.88.2.2	config	805
8.88.2.3	cardmode	805
8.88.2.4	errors	805
8.88.2.5	rca	805
8.89	SDCDriverVMT Struct Reference	805
8.89.1	Detailed Description	806
8.90	SerialConfig Struct Reference	806
8.90.1	Detailed Description	807
8.90.2	Field Documentation	807
8.90.2.1	speed	807
8.91	SerialDriver Struct Reference	807
8.91.1	Detailed Description	808
8.91.2	Field Documentation	808
8.91.2.1	vmt	808
8.92	SerialDriverVMT Struct Reference	808
8.92.1	Detailed Description	810
8.93	SerialUSBConfig Struct Reference	810
8.93.1	Detailed Description	811
8.93.2	Field Documentation	811
8.93.2.1	usbp	811
8.93.2.2	bulk_in	811
8.93.2.3	bulk_out	811
8.93.2.4	int_in	811
8.94	SerialUSBDriver Struct Reference	811
8.94.1	Detailed Description	813
8.94.2	Field Documentation	813
8.94.2.1	vmt	813
8.95	SerialUSBDriverVMT Struct Reference	813
8.95.1	Detailed Description	815
8.96	SHA1Context Struct Reference	815
8.96.1	Detailed Description	815
8.97	SHA256Context Struct Reference	815
8.97.1	Detailed Description	815
8.98	SHA512Context Struct Reference	816
8.98.1	Detailed Description	816
8.99	SPIConfig Struct Reference	816

8.99.1 Detailed Description	817
8.99.2 Field Documentation	817
8.99.2.1 circular	817
8.99.2.2 end_cb	817
8.99.2.3 ssline	817
8.99.2.4 ssport	817
8.99.2.5 ssmask	817
8.99.2.6 sspad	817
8.100SPIDriver Struct Reference	818
8.100.1 Detailed Description	818
8.100.2 Field Documentation	818
8.100.2.1 state	819
8.100.2.2 config	819
8.100.2.3 thread	819
8.100.2.4 mutex	819
8.101threads_queue_t Struct Reference	819
8.101.1 Detailed Description	819
8.102UARTConfig Struct Reference	819
8.102.1 Detailed Description	820
8.102.2 Field Documentation	820
8.102.2.1 txend1_cb	821
8.102.2.2 txend2_cb	821
8.102.2.3 rxend_cb	821
8.102.2.4 rxchar_cb	821
8.102.2.5 rxerr_cb	821
8.103UARTDriver Struct Reference	821
8.103.1 Detailed Description	822
8.103.2 Field Documentation	823
8.103.2.1 state	823
8.103.2.2 txstate	823
8.103.2.3 rxstate	823
8.103.2.4 config	823
8.103.2.5 early	823
8.103.2.6 threadrx	823
8.103.2.7 threadtx	823
8.103.2.8 mutex	823
8.104unpacked_mmc_cid_t Struct Reference	823
8.104.1 Detailed Description	824
8.105unpacked_mmc_csd_t Struct Reference	824
8.105.1 Detailed Description	824

8.106unpacked_sdc_cid_t Struct Reference	825
8.106.1 Detailed Description	825
8.107unpacked_sdc_csd_10_t Struct Reference	825
8.107.1 Detailed Description	826
8.108unpacked_sdc_csd_20_t Struct Reference	826
8.108.1 Detailed Description	827
8.109USBConfig Struct Reference	827
8.109.1 Detailed Description	829
8.109.2 Field Documentation	829
8.109.2.1 event_cb	829
8.109.2.2 get_descriptor_cb	829
8.109.2.3 requests_hook_cb	829
8.109.2.4 sof_cb	829
8.110USBDescriptor Struct Reference	829
8.110.1 Detailed Description	830
8.110.2 Field Documentation	830
8.110.2.1 ud_size	830
8.110.2.2 ud_string	830
8.111USBDriver Struct Reference	830
8.111.1 Detailed Description	831
8.111.2 Field Documentation	831
8.111.2.1 state	831
8.111.2.2 config	831
8.111.2.3 transmitting	831
8.111.2.4 receiving	831
8.111.2.5 epc	832
8.111.2.6 in_params	832
8.111.2.7 out_params	832
8.111.2.8 ep0state	832
8.111.2.9 ep0next	832
8.111.2.10ep0n	832
8.111.2.11ep0endcb	832
8.111.2.12setup	832
8.111.2.13status	832
8.111.2.14address	832
8.111.2.15configuration	832
8.111.2.16saved_state	833
8.112USBEndpointConfig Struct Reference	833
8.112.1 Detailed Description	835
8.112.2 Field Documentation	835

8.112.2.1 ep_mode	835
8.112.2.2 setup_cb	835
8.112.2.3 in_cb	835
8.112.2.4 out_cb	835
8.112.2.5 in_maxsize	836
8.112.2.6 out_maxsize	836
8.112.2.7 in_state	836
8.112.2.8 out_state	836
8.113USBInEndpointState Struct Reference	836
8.113.1 Detailed Description	837
8.113.2 Field Documentation	837
8.113.2.1 txsize	837
8.113.2.2 txcnt	837
8.113.2.3 txbuf	837
8.113.2.4 thread	837
8.114USBOutEndpointState Struct Reference	837
8.114.1 Detailed Description	838
8.114.2 Field Documentation	838
8.114.2.1 rxsize	838
8.114.2.2 rxcnt	838
8.114.2.3 rxbuf	838
8.114.2.4 thread	838
8.115WDGConfig Struct Reference	838
8.115.1 Detailed Description	839
8.116WDGDriver Struct Reference	839
8.116.1 Detailed Description	839
8.116.2 Field Documentation	839
8.116.2.1 state	839
8.116.2.2 config	840
9 File Documentation	841
9.1 hal.c File Reference	841
9.1.1 Detailed Description	841
9.2 hal.h File Reference	841
9.2.1 Detailed Description	843
9.3 hal_accelerometer.h File Reference	843
9.3.1 Detailed Description	843
9.4 hal_adc.c File Reference	844
9.4.1 Detailed Description	844
9.5 hal_adc.h File Reference	844

9.5.1 Detailed Description	845
9.6 hal_adc_ild.c File Reference	846
9.6.1 Detailed Description	846
9.7 hal_adc_ild.h File Reference	846
9.7.1 Detailed Description	847
9.8 hal_barometer.h File Reference	847
9.8.1 Detailed Description	848
9.9 hal_buffers.c File Reference	848
9.9.1 Detailed Description	849
9.10 hal_buffers.h File Reference	849
9.10.1 Detailed Description	851
9.11 hal_can.c File Reference	851
9.11.1 Detailed Description	852
9.12 hal_can.h File Reference	852
9.12.1 Detailed Description	853
9.13 hal_can_ild.c File Reference	854
9.13.1 Detailed Description	854
9.14 hal_can_ild.h File Reference	854
9.14.1 Detailed Description	855
9.15 hal_channels.h File Reference	855
9.15.1 Detailed Description	857
9.16 hal_compass.h File Reference	857
9.16.1 Detailed Description	858
9.17 hal_crypto.c File Reference	858
9.17.1 Detailed Description	859
9.18 hal_crypto.h File Reference	860
9.18.1 Detailed Description	862
9.19 hal_crypto_ild.c File Reference	862
9.19.1 Detailed Description	864
9.20 hal_crypto_ild.h File Reference	864
9.20.1 Detailed Description	866
9.21 hal_dac.c File Reference	866
9.21.1 Detailed Description	867
9.22 hal_dac.h File Reference	867
9.22.1 Detailed Description	868
9.23 hal_dac_ild.c File Reference	868
9.23.1 Detailed Description	869
9.24 hal_dac_ild.h File Reference	869
9.24.1 Detailed Description	870
9.25 hal_ext.c File Reference	870

9.25.1 Detailed Description	871
9.26 hal_ext.h File Reference	871
9.26.1 Detailed Description	872
9.27 hal_ext_ild.c File Reference	872
9.27.1 Detailed Description	872
9.28 hal_ext_ild.h File Reference	873
9.28.1 Detailed Description	873
9.29 hal_files.h File Reference	873
9.29.1 Detailed Description	874
9.30 hal_flash.c File Reference	875
9.30.1 Detailed Description	875
9.31 hal_flash.h File Reference	875
9.31.1 Detailed Description	876
9.32 hal_gpt.c File Reference	876
9.32.1 Detailed Description	877
9.33 hal_gpt.h File Reference	877
9.33.1 Detailed Description	878
9.34 hal_gpt_ild.c File Reference	878
9.34.1 Detailed Description	879
9.35 hal_gpt_ild.h File Reference	879
9.35.1 Detailed Description	880
9.36 hal_gyroscope.h File Reference	880
9.36.1 Detailed Description	881
9.37 hal_hygrometer.h File Reference	881
9.37.1 Detailed Description	882
9.38 hal_i2c.c File Reference	882
9.38.1 Detailed Description	882
9.39 hal_i2c.h File Reference	882
9.39.1 Detailed Description	884
9.40 hal_i2c_ild.c File Reference	884
9.40.1 Detailed Description	884
9.41 hal_i2c_ild.h File Reference	884
9.41.1 Detailed Description	885
9.42 hal_i2s.c File Reference	885
9.42.1 Detailed Description	886
9.43 hal_i2s.h File Reference	886
9.43.1 Detailed Description	887
9.44 hal_i2s_ild.c File Reference	887
9.44.1 Detailed Description	887
9.45 hal_i2s_ild.h File Reference	887

9.45.1 Detailed Description	888
9.46 hal_icu.c File Reference	888
9.46.1 Detailed Description	889
9.47 hal_icu.h File Reference	889
9.47.1 Detailed Description	890
9.48 hal_icu_ild.c File Reference	890
9.48.1 Detailed Description	891
9.49 hal_icu_ild.h File Reference	891
9.49.1 Detailed Description	892
9.50 hal_joblock.h File Reference	892
9.50.1 Detailed Description	893
9.51 hal_jesd216_flash.c File Reference	893
9.51.1 Detailed Description	893
9.52 hal_jesd216_flash.h File Reference	893
9.52.1 Detailed Description	894
9.53 hal_ild.c File Reference	894
9.53.1 Detailed Description	894
9.54 hal_ild.h File Reference	894
9.54.1 Detailed Description	895
9.55 hal_mac.c File Reference	895
9.55.1 Detailed Description	895
9.56 hal_mac.h File Reference	895
9.56.1 Detailed Description	896
9.57 hal_mac_ild.c File Reference	897
9.57.1 Detailed Description	897
9.58 hal_mac_ild.h File Reference	897
9.58.1 Detailed Description	898
9.59 hal_mii.h File Reference	899
9.59.1 Detailed Description	901
9.60 hal_mmc_spi.c File Reference	901
9.60.1 Detailed Description	902
9.61 hal_mmc_spi.h File Reference	902
9.61.1 Detailed Description	903
9.62 hal_mmcsd.c File Reference	903
9.62.1 Detailed Description	904
9.63 hal_mmcsd.h File Reference	904
9.63.1 Detailed Description	908
9.64 hal_objects.h File Reference	908
9.64.1 Detailed Description	909
9.65 hal_pal.c File Reference	909

9.65.1 Detailed Description	909
9.66 hal_pal.h File Reference	909
9.66.1 Detailed Description	913
9.67 hal_pal_ild.c File Reference	913
9.67.1 Detailed Description	913
9.68 hal_pal_ild.h File Reference	913
9.68.1 Detailed Description	915
9.69 hal_pwm.c File Reference	915
9.69.1 Detailed Description	915
9.70 hal_pwm.h File Reference	916
9.70.1 Detailed Description	917
9.71 hal_pwm_ild.c File Reference	917
9.71.1 Detailed Description	918
9.72 hal_pwm_ild.h File Reference	918
9.72.1 Detailed Description	919
9.73 hal_qspi.c File Reference	919
9.73.1 Detailed Description	920
9.74 hal_qspi.h File Reference	920
9.74.1 Detailed Description	922
9.75 hal_qspi_ild.c File Reference	922
9.75.1 Detailed Description	923
9.76 hal_qspi_ild.h File Reference	923
9.76.1 Detailed Description	924
9.77 hal_queues.c File Reference	924
9.77.1 Detailed Description	925
9.78 hal_queues.h File Reference	925
9.78.1 Detailed Description	927
9.79 hal_rtc.c File Reference	927
9.79.1 Detailed Description	927
9.80 hal_rtc.h File Reference	928
9.80.1 Detailed Description	929
9.81 hal_rtc_ild.c File Reference	929
9.81.1 Detailed Description	929
9.82 hal_rtc_ild.h File Reference	929
9.82.1 Detailed Description	930
9.83 hal_sdc.c File Reference	930
9.83.1 Detailed Description	932
9.84 hal_sdc.h File Reference	932
9.84.1 Detailed Description	933
9.85 hal_sdc_ild.c File Reference	933

9.85.1 Detailed Description	934
9.86 hal_sdc_ild.h File Reference	934
9.86.1 Detailed Description	936
9.87 hal_sensors.h File Reference	936
9.87.1 Detailed Description	936
9.88 hal_serial.c File Reference	936
9.88.1 Detailed Description	937
9.89 hal_serial.h File Reference	937
9.89.1 Detailed Description	939
9.90 hal_serial_ild.c File Reference	939
9.90.1 Detailed Description	939
9.91 hal_serial_ild.h File Reference	939
9.91.1 Detailed Description	940
9.92 hal_serial_usb.c File Reference	940
9.92.1 Detailed Description	941
9.93 hal_serial_usb.h File Reference	941
9.93.1 Detailed Description	942
9.94 hal_spi.c File Reference	942
9.94.1 Detailed Description	943
9.95 hal_spi.h File Reference	943
9.95.1 Detailed Description	945
9.96 hal_spi_ild.c File Reference	945
9.96.1 Detailed Description	946
9.97 hal_spi_ild.h File Reference	946
9.97.1 Detailed Description	947
9.98 hal_st.c File Reference	947
9.98.1 Detailed Description	948
9.99 hal_st.h File Reference	948
9.99.1 Detailed Description	948
9.100 hal_st_ild.c File Reference	948
9.100.1 Detailed Description	949
9.101 hal_st_ild.h File Reference	949
9.101.1 Detailed Description	949
9.102 hal_streams.h File Reference	949
9.102.1 Detailed Description	950
9.103 hal_thermometer.h File Reference	950
9.103.1 Detailed Description	951
9.104 hal_uart.c File Reference	951
9.104.1 Detailed Description	952
9.105 hal_uart.h File Reference	952

9.105.1 Detailed Description	954
9.106hal_uart_lld.c File Reference	954
9.106.1 Detailed Description	954
9.107hal_uart_lld.h File Reference	954
9.107.1 Detailed Description	955
9.108hal_usb.c File Reference	956
9.108.1 Detailed Description	957
9.109hal_usb.h File Reference	957
9.109.1 Detailed Description	959
9.110hal_usb_cdc.h File Reference	960
9.110.1 Detailed Description	961
9.111hal_usb_lld.c File Reference	961
9.111.1 Detailed Description	962
9.111.2 Variable Documentation	962
9.111.2.1 in	962
9.111.2.2 out	962
9.112hal_usb_lld.h File Reference	962
9.112.1 Detailed Description	964
9.113hal_wdg.c File Reference	964
9.113.1 Detailed Description	964
9.114hal_wdg.h File Reference	964
9.114.1 Detailed Description	965
9.115hal_wdg_lld.c File Reference	965
9.115.1 Detailed Description	965
9.116hal_wdg_lld.h File Reference	965
9.116.1 Detailed Description	966
9.117halconf.h File Reference	966
9.117.1 Detailed Description	969
9.118mfs.c File Reference	969
9.118.1 Detailed Description	970
9.119mfs.h File Reference	970
9.119.1 Detailed Description	971
9.120osal.c File Reference	971
9.120.1 Detailed Description	972
9.121osal.h File Reference	973
9.121.1 Detailed Description	976

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

Global `canReceive` (canp, mailbox, crfp, timeout)

Global `canTransmit` (canp, mailbox, ctfp, timeout)

Global `sdGetWouldBlock` (`SerialDriver` *sdp)

Global `sdPutWouldBlock` (`SerialDriver` *sdp)

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

HAL	323
Configuration	326
Normal Drivers	335
ADC Driver	28
CAN Driver	50
Crypto Driver	73
DAC Driver	127
EXT Driver	145
GPT Driver	164
HAL Driver	184
I2C Driver	264
I2S Driver	278
ICU Driver	287
MAC Driver	307
PAL Driver	425
PWM Driver	466
QSPI Driver	489
RTC Driver	508
SDC Driver	518
Serial Driver	549
SPI Driver	576
ST Driver	604
UART Driver	615
USB Driver	643
WDG Driver	685
Complex Drivers	337
Managed Flash Storage Driver	365
MMC over SPI Driver	393
Serial over USB Driver	565
Interfaces and Classes	338
Abstract I/O Channel	210
Abstract Files	219
Abstract I/O Block Device	225
Base Object	232
Abstract Peripheral Interfaces	235
Generic Accelerometer Interface	23
Generic Barometer Interface	45

Generic Compass Interface	68
Generic Flash Driver Interface	156
Generic Gyroscope Interface	179
Generic Hygrometer Interface	259
JESD216 Compliant Flash Driver Interface	305
Generic Sensor Interface	546
Generic Thermometer Interface	610
Abstract Streams	256
Inner Code	339
I/O Buffers Queues	187
I/O Bytes Queues	236
MMC/SD Block Device	414
Support Code	340
MII/RMII Header	383
USB CDC Header	683
OSAL	341

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

ADCCConfig	691
ADCConversionGroup	692
ADCDriver	693
BaseAccelerometerVMT	697
BaseBarometerVMT	704
BaseBlockDeviceVMT	707
MMCSDBlockDeviceVMT	787
MMCDriverVMT	784
SDCDriverVMT	805
BaseCompassVMT	713
BaseFlashVMT	716
BaseGyroscopeVMT	718
BaseHygrometerVMT	720
BaseObject	721
BaseBlockDevice	705
MMCSDBlockDevice	785
MMCDriver	782
BaseFlash	714
MFSDriver	777
BaseSensor	723
BaseAccelerometer	695
BaseBarometer	702
BaseCompass	712
BaseGyroscope	717
BaseHygrometer	719
BaseThermometer	728
BaseSequentialStream	725
BaseChannel	708
BaseAsynchronousChannel	698
SerialDriver	807
SerialUSBDriver	811
FileStream	749
RTCDriverVMT	800
BaseObjectVMT	722
BaseSensorVMT	724
BaseSequentialStreamVMT	727

BaseChannelVMT	710
BaseAsynchronousChannelVMT	700
SerialDriverVMT	808
SerialUSBDriverVMT	813
FileStreamVMT	750
BaseThermometerVMT	730
BlockDeviceInfo	731
CANConfig	732
CANDriver	732
CANRxFrame	735
CANTxFrame	737
cdc_linecoding_t	738
CRYConfig	739
CRYDriver	739
DACConfig	741
DACConversionGroup	741
DACDriver	743
event_source	744
EXTChannelConfig	745
EXTConfig	747
EXTDriver	748
flash_descriptor_t	751
flash_sector_descriptor_t	753
GPTConfig	754
GPTDriver	755
I2CConfig	756
I2CDriver	757
I2SConfig	758
I2SDriver	759
ICUConfig	760
ICUDriver	762
io_buffers_queue	763
io_queue	765
IOBus	767
JESD215Flash	769
JESD215FlashVMT	769
MACConfig	770
MACDriver	771
MACReceiveDescriptor	772
MACTransmitDescriptor	773
mfs_bank_header_t	774
mfs_data_header_t	775
MFSCConfig	776
MMCCConfig	781
PALConfig	789
palevent_t	789
PWMChannelConfig	791
PWMConfig	792
PWMDriver	793
qspi_command_t	795
QSPICConfig	795
QSPIDriver	796
RTCAAlarm	798
RTCDateTime	798
RTCDriver	800
SDCConfig	802
SDCDriver	803
SerialConfig	806

SerialUSBConfig	810
SHA1Context	815
SHA256Context	815
SHA512Context	816
SPIConfig	816
SPIDriver	818
threads_queue_t	819
UARTConfig	819
UARTDriver	821
unpacked_mmc_cid_t	823
unpacked_mmc_csd_t	824
unpacked_sdc_cid_t	825
unpacked_sdc_csd_10_t	825
unpacked_sdc_csd_20_t	826
USBConfig	827
USBDescriptor	829
USBDriver	830
USBEndpointConfig	833
USBInEndpointState	836
USBOutEndpointState	837
WDGConfig	838
WDGDriver	839

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

ADConfig	Driver configuration structure	691
ADCConversionGroup	Conversion group configuration structure	692
ADCDriver	Structure representing an ADC driver	693
BaseAccelerometer	Base accelerometer class	695
BaseAccelerometerVMT	BaseAccelerometer virtual methods table	697
BaseAsynchronousChannel	Base asynchronous channel class	698
BaseAsynchronousChannelVMT	BaseAsynchronousChannel virtual methods table	700
BaseBarometer	Base barometer class	702
BaseBarometerVMT	BaseBarometer virtual methods table	704
BaseBlockDevice	Base block device class	705
BaseBlockDeviceVMT	BaseBlockDevice virtual methods table	707
BaseChannel	Base channel class	708
BaseChannelVMT	BaseChannel virtual methods table	710
BaseCompass	Base compass class	712
BaseCompassVMT	BaseCompass virtual methods table	713
BaseFlash	Base flash class	714
BaseFlashVMT	BaseFlash virtual methods table	716
BaseGyroscope	Base gyroscope class	717
BaseGyroscopeVMT	BaseGyroscope virtual methods table	718

BaseHygrometer	
Base hygrometer class	719
BaseHygrometerVMT	
BaseHygrometer virtual methods table	720
BaseObject	
Base stream class	721
BaseObjectVMT	
BaseObject virtual methods table	722
BaseSensor	
Base stream class	723
BaseSensorVMT	
BaseSensor virtual methods table	724
BaseSequentialStream	
Base stream class	725
BaseSequentialStreamVMT	
BaseSequentialStream virtual methods table	727
BaseThermometer	
Base thermometer class	728
BaseThermometerVMT	
BaseThermometer virtual methods table	730
BlockDeviceInfo	
Block device info	731
CANConfig	
Driver configuration structure	732
CANDriver	
Structure representing an CAN driver	732
CANRxFrame	
CAN received frame	735
CANTxFrame	
CAN transmission frame	737
cdc_linecoding_t	
Type of Line Coding structure	738
CRYConfig	
Driver configuration structure	739
CRYDriver	
Structure representing an CRY driver	739
DACConfig	
Driver configuration structure	741
DACConversionGroup	
DAC Conversion group structure	741
DACDriver	
Structure representing a DAC driver	743
event_source	
Events source object	744
EXTChannelConfig	
Channel configuration structure	745
EXTConfig	
Driver configuration structure	747
EXTDriver	
Structure representing an EXT driver	748
FileStream	
Base file stream class	749
FileStreamVMT	
FileStream virtual methods table	750
flash_descriptor_t	
Type of a flash device descriptor	751
flash_sector_descriptor_t	
Flash sector descriptor	753

GPTConfig	Driver configuration structure	754
GPTDriver	Structure representing a GPT driver	755
I2CConfig	Type of I2C driver configuration structure	756
I2CDriver	Structure representing an I2C driver	757
I2SConfig	Driver configuration structure	758
I2SDriver	Structure representing an I2S driver	759
ICUConfig	Driver configuration structure	760
ICUDriver	Structure representing an ICU driver	762
io_buffers_queue	Structure of a generic buffers queue	763
io_queue	Generic I/O queue structure	765
IOBus	I/O bus descriptor	767
JESD215Flash	Base flash class	769
JESD215FlashVMT	JESD215Flash virtual methods table	769
MACConfig	Driver configuration structure	770
MACDriver	Structure representing a MAC driver	771
MACReceiveDescriptor	Structure representing a receive descriptor	772
MACTransmitDescriptor	Structure representing a transmit descriptor	773
mfs_bank_header_t	Type of a bank header	774
mfs_data_header_t	Type of a data block header	775
MFSCConfig	Type of a MFS configuration structure	776
MFSDriver	Type of an MFS instance	777
MMCCConfig	MMC/SD over SPI driver configuration structure	781
MMCDriver	Structure representing a MMC/SD over SPI driver	782
MMCDriverVMT	MMCDriver virtual methods table	784
MMCSDBlockDevice	MCC/SD block device class	785
MMCSDBlockDeviceVMT	MMCSDBlockDevice virtual methods table	787
PALConfig	Generic I/O ports static initializer	789
palevent_t	Type of a PAL event record	789
PWMChannelConfig	Type of a PWM driver channel configuration structure	791

PWMConfig	Type of a PWM driver configuration structure	792
PWMDriver	Structure representing a PWM driver	793
qspi_command_t	Type of a QSPI command descriptor	795
QSPIConfig	Driver configuration structure	795
QSPIDriver	Structure representing an QSPI driver	796
RTCAAlarm	Type of a structure representing an RTC alarm time stamp	798
RTCDateTime	Type of a structure representing an RTC date/time stamp	798
RTCDriver	Structure representing an RTC driver	800
RTCDriverVMT	RTCDriver virtual methods table	800
SDCConfig	Driver configuration structure	802
SDCDriver	Structure representing an SDC driver	803
SDCDriverVMT	SDCDriver virtual methods table	805
SerialConfig	PLATFORM Serial Driver configuration structure	806
SerialDriver	Full duplex serial driver class	807
SerialDriverVMT	SerialDriver virtual methods table	808
SerialUSBConfig	Serial over USB Driver configuration structure	810
SerialUSBDriver	Full duplex serial driver class	811
SerialUSBDriverVMT	SerialDriver virtual methods table	813
SHA1Context	Type of a SHA1 context	815
SHA256Context	Type of a SHA256 context	815
SHA512Context	Type of a SHA512 context	816
SPICConfig	Driver configuration structure	816
SPIDriver	Structure representing an SPI driver	818
threads_queue_t	Type of a thread queue	819
UARTConfig	Driver configuration structure	819
UARTDriver	Structure representing an UART driver	821
unpacked_mmc_cid_t	Unpacked CID register from MMC	823
unpacked_mmc_csd_t	Unpacked CSD register from MMC	824
unpacked_sdc_cid_t	Unpacked CID register from SDC	825

unpacked_sdc_csd_10_t	
Unpacked CSD v1.0 register from SDC	825
unpacked_sdc_csd_20_t	
Unpacked CSD v2.0 register from SDC	826
USBConfig	
Type of an USB driver configuration structure	827
USBDescriptor	
Type of an USB descriptor	829
USBDriver	
Structure representing an USB driver	830
USBEndpointConfig	
Type of an USB endpoint configuration structure	833
USBInEndpointState	
Type of an IN endpoint state structure	836
USBOutEndpointState	
Type of an OUT endpoint state structure	837
WDGConfig	
Driver configuration structure	838
WDGDriver	
Structure representing an WDG driver	839

Chapter 6

File Index

6.1 File List

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

hal.c	HAL subsystem code	841
hal.h	HAL subsystem header	841
hal_accelerometer.h	Generic accelerometer interface header	843
hal_adc.c	ADC Driver code	844
hal_adc.h	ADC Driver macros and structures	844
hal_adc_ild.c	PLATFORM ADC subsystem low level driver source	846
hal_adc_ild.h	PLATFORM ADC subsystem low level driver header	846
hal_barometer.h	Generic barometer interface header	847
hal_buffers.c	I/O Buffers code	848
hal_buffers.h	I/O Buffers macros and structures	849
hal_can.c	CAN Driver code	851
hal_can.h	CAN Driver macros and structures	852
hal_can_ild.c	PLATFORM CAN subsystem low level driver source	854
hal_can_ild.h	PLATFORM CAN subsystem low level driver header	854
hal_channels.h	I/O channels access	855
hal_compass.h	Generic compass interface header	857
hal_crypto.c	Cryptographic Driver code	858
hal_crypto.h	Cryptographic Driver macros and structures	860
hal_crypto_ild.c	PLATFORM cryptographic subsystem low level driver source	862

hal_crypto_lld.h	PLATFORM cryptographic subsystem low level driver header	864
hal_dac.c	DAC Driver code	866
hal_dac.h	DAC Driver macros and structures	867
hal_dac_lld.c	PLATFORM DAC subsystem low level driver source	868
hal_dac_lld.h	PLATFORM DAC subsystem low level driver header	869
hal_ext.c	EXT Driver code	870
hal_ext.h	EXT Driver macros and structures	871
hal_ext_lld.c	PLATFORM EXT subsystem low level driver source	872
hal_ext_lld.h	PLATFORM EXT subsystem low level driver header	873
hal_files.h	Data files	873
hal_flash.c	Generic flash driver class code	875
hal_flash.h	Generic flash driver class header	875
hal_gpt.c	GPT Driver code	876
hal_gpt.h	GPT Driver macros and structures	877
hal_gpt_lld.c	PLATFORM GPT subsystem low level driver source	878
hal_gpt_lld.h	PLATFORM GPT subsystem low level driver header	879
hal_gyroscope.h	Generic gyroscope interface header	880
hal_hygrometer.h	Generic hygrometer interface header	881
hal_i2c.c	I2C Driver code	882
hal_i2c.h	I2C Driver macros and structures	882
hal_i2c_lld.c	PLATFORM I2C subsystem low level driver source	884
hal_i2c_lld.h	PLATFORM I2C subsystem low level driver header	884
hal_i2s.c	I2S Driver code	885
hal_i2s.h	I2S Driver macros and structures	886
hal_i2s_lld.c	PLATFORM I2S subsystem low level driver source	887
hal_i2s_lld.h	PLATFORM I2S subsystem low level driver header	887
hal_icu.c	ICU Driver code	888
hal_icu.h	ICU Driver macros and structures	889
hal_icu_lld.c	PLATFORM ADC subsystem low level driver source	890

hal_icu_ild.h	PLATFORM ICU subsystem low level driver header	891
hal_ioblock.h	I/O block devices access	892
hal_jesd216_flash.c	JESD216 compliant flash driver class code	893
hal_jesd216_flash.h	JESD216 compliant flash driver class header	893
hal_ild.c	PLATFORM HAL subsystem low level driver source	894
hal_ild.h	PLATFORM HAL subsystem low level driver header	894
hal_mac.c	MAC Driver code	895
hal_mac.h	MAC Driver macros and structures	895
hal_mac_ild.c	PLATFORM MAC subsystem low level driver source	897
hal_mac_ild.h	PLATFORM MAC subsystem low level driver header	897
hal_mii.h	MII macros and structures	899
hal_mmc_spi.c	MMC over SPI driver code	901
hal_mmc_spi.h	MMC over SPI driver header	902
hal_mmcscd.c	MMC/SD cards common code	903
hal_mmcscd.h	MMC/SD cards common header	904
hal_objects.h	Base object	908
hal_pal.c	I/O Ports Abstraction Layer code	909
hal_pal.h	I/O Ports Abstraction Layer macros, types and structures	909
hal_pal_ild.c	PLATFORM PAL subsystem low level driver source	913
hal_pal_ild.h	PLATFORM PAL subsystem low level driver header	913
hal_pwm.c	PWM Driver code	915
hal_pwm.h	PWM Driver macros and structures	916
hal_pwm_ild.c	PLATFORM PWM subsystem low level driver source	917
hal_pwm_ild.h	PLATFORM PWM subsystem low level driver header	918
hal_qspi.c	QSPI Driver code	919
hal_qspi.h	QSPI Driver macros and structures	920
hal_qspi_ild.c	PLATFORM QSPI subsystem low level driver source	922
hal_qspi_ild.h	PLATFORM QSPI subsystem low level driver header	923
hal_queues.c	I/O Queues code	924

hal_queues.h	I/O Queues macros and structures	925
hal_rtc.c	RTC Driver code	927
hal_rtc.h	RTC Driver macros and structures	928
hal_rtc_lld.c	PLATFORM RTC subsystem low level driver source	929
hal_rtc_lld.h	PLATFORM RTC subsystem low level driver header	929
hal_sdc.c	SDC Driver code	930
hal_sdc.h	SDC Driver macros and structures	932
hal_sdc_lld.c	PLATFORM SDC subsystem low level driver source	933
hal_sdc_lld.h	PLATFORM SDC subsystem low level driver header	934
hal_sensors.h	Generic sensors interface header	936
hal_serial.c	Serial Driver code	936
hal_serial.h	Serial Driver macros and structures	937
hal_serial_lld.c	PLATFORM serial subsystem low level driver source	939
hal_serial_lld.h	PLATFORM serial subsystem low level driver header	939
hal_serial_usb.c	Serial over USB Driver code	940
hal_serial_usb.h	Serial over USB Driver macros and structures	941
hal_spi.c	SPI Driver code	942
hal_spi.h	SPI Driver macros and structures	943
hal_spi_lld.c	PLATFORM SPI subsystem low level driver source	945
hal_spi_lld.h	PLATFORM SPI subsystem low level driver header	946
hal_st.c	ST Driver code	947
hal_st.h	ST Driver macros and structures	948
hal_st_lld.c	PLATFORM ST subsystem low level driver source	948
hal_st_lld.h	PLATFORM ST subsystem low level driver header	949
hal_streams.h	Data streams	949
hal_thermometer.h	Generic thermometer interface header	950
hal_uart.c	UART Driver code	951
hal_uart.h	UART Driver macros and structures	952
hal_uart_lld.c	PLATFORM UART subsystem low level driver source	954

hal_uart_lld.h	PLATFORM UART subsystem low level driver header	954
hal_usb.c	USB Driver code	956
hal_usb.h	USB Driver macros and structures	957
hal_usb_cdc.h	USB CDC macros and structures	960
hal_usb_lld.c	PLATFORM USB subsystem low level driver source	961
hal_usb_lld.h	PLATFORM USB subsystem low level driver header	962
hal_wdg.c	WDG Driver code	964
hal_wdg.h	WDG Driver macros and structures	964
hal_wdg_lld.c	WDG Driver subsystem low level driver source template	965
hal_wdg_lld.h	WDG Driver subsystem low level driver header template	965
halconf.h	HAL configuration header	966
mfs.c	Managed Flash Storage module code	969
mfs.h	Managed Flash Storage module header	970
osal.c	OSAL module code	971
osal.h	OSAL module header	973

Chapter 7

Module Documentation

7.1 Generic Accelerometer Interface

HAL Generic Accelerometer Interface.

7.1.1 Detailed Description

HAL Generic Accelerometer Interface.

Macros

- #define `_base_accelerometer_methods_alone`
BaseAccelerometer specific methods.
- #define `_base_accelerometer_methods`
BaseAccelerometer specific methods with inherited ones.
- #define `_base_accelerometer_data_base_sensor_data`
BaseAccelerometer specific data.

Macro Functions (BaseAccelerometer)

- #define `accelerometerGetAxesNumber(ip)` (ip)->vmt->get_channels_number(ip)
Accelerometer get axes number.
- #define `accelerometerReadRaw(ip, dp)` (ip)->vmt->read_raw(ip, dp)
Accelerometer read raw data.
- #define `accelerometerReadCooked(ip, dp)` (ip)->vmt->read_cooked(ip, dp)
Accelerometer read cooked data.
- #define `accelerometerSetBias(ip, bp)` (ip)->vmt->set_bias(ip, bp)
Updates accelerometer bias data from received buffer.
- #define `accelerometerResetBias(ip)` (ip)->vmt->reset_bias(ip)
Reset accelerometer bias data restoring it to zero.
- #define `accelerometerSetSensitivity(ip, sp)` (ip)->vmt->set_sensitivity(ip, sp)
Updates accelerometer sensitivity data from received buffer.
- #define `accelerometerResetSensitivity(ip)` (ip)->vmt->reset_sensitivity(ip)
Reset accelerometer sensitivity data restoring it to its typical value.

Data Structures

- struct [BaseAccelerometerVMT](#)
BaseAccelerometer virtual methods table.
- struct [BaseAccelerometer](#)
Base accelerometer class.

7.1.2 Macro Definition Documentation

7.1.2.1 #define _base_accelerometer_methods_alone

Value:

```
/* Invoke the set bias procedure.*/
msg_t (*set_bias)(void *instance, float biases[]);
/* Remove bias stored data.*/
msg_t (*reset_bias)(void *instance);
/* Invoke the set sensitivity procedure.*/
msg_t (*set_sensitivity)(void *instance, float sensitivities[]);
/* Restore sensitivity stored data to default.*/
msg_t (*reset_sensitivity)(void *instance);
```

[BaseAccelerometer](#) specific methods.

7.1.2.2 #define _base_accelerometer_methods

Value:

```
_base_sensor_methods
_base_accelerometer_methods_alone
```

[BaseAccelerometer](#) specific methods with inherited ones.

7.1.2.3 #define _base_accelerometer_data _base_sensor_data

[BaseAccelerometer](#) specific data.

7.1.2.4 #define accelerometerGetAxesNumber(ip)(ip)->vmt->get_channels_number(ip)

Accelerometer get axes number.

Parameters

in	<i>ip</i>	pointer to a BaseAccelerometer class.
----	-----------	---

Returns

The number of axes of the [BaseAccelerometer](#)

Function Class:

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

7.1.2.5 #define accelerometerReadRaw(ip, dp)(ip)->vmt->read_raw(ip, dp)

Accelerometer read raw data.

Parameters

in	<i>ip</i>	pointer to a BaseAccelerometer class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.1.2.6 `#define accelerometerReadCooked(ip, dp) (ip)->vmt->read_cooked(ip, dp)`

Accelerometer read cooked data.

Parameters

in	<i>ip</i>	pointer to a BaseAccelerometer class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.1.2.7 `#define accelerometerSetBias(ip, bp) (ip)->vmt->set_bias(ip, bp)`

Updates accelerometer bias data from received buffer.

Note

The bias buffer must have the same length of the the accelerometer axes number.

Parameters

in	<i>ip</i>	pointer to a BaseAccelerometer class.
in	<i>bp</i>	pointer to a buffer of bias values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.1.2.8 `#define accelerometerResetBias(ip)(ip)->vmt->reset_bias(ip)`

Reset accelerometer bias data restoring it to zero.

Parameters

in	<i>ip</i>	pointer to a BaseAccelerometer class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.1.2.9 `#define accelerometerSetSensitivity(ip, sp)(ip)->vmt->set_sensitivity(ip, sp)`

Updates accelerometer sensitivity data from received buffer.

Note

The sensitivity buffer must have the same length of the the accelerometer axes number.

Parameters

in	<i>ip</i>	pointer to a BaseAccelerometer class.
in	<i>sp</i>	pointer to a buffer of sensitivity values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.1.2.10 `#define accelerometerResetSensitivity(ip) (ip)->vmt->reset_sensitivity(ip)`

Reset accelerometer sensitivity data restoring it to its typical value.

Parameters

in	<i>ip</i>	pointer to a BaseAccelerometer class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.2 ADC Driver

Generic ADC Driver.

7.2.1 Detailed Description

Generic ADC Driver.

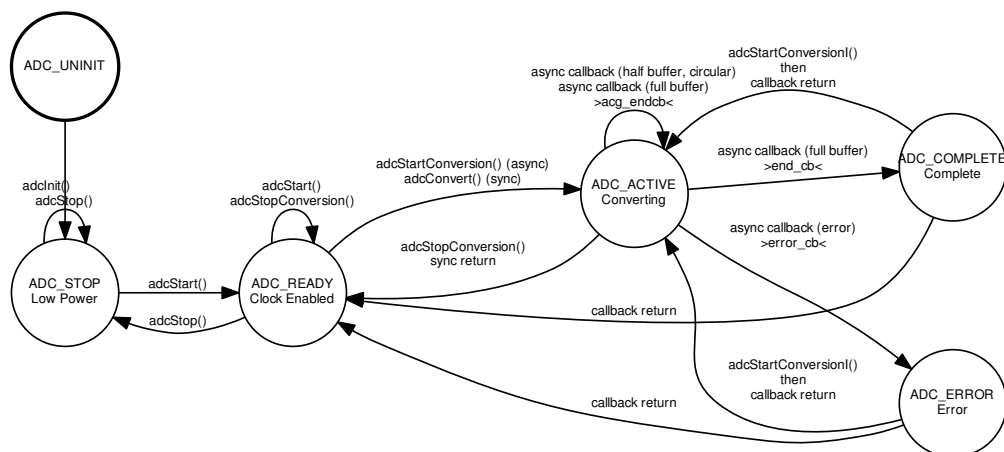
This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

Precondition

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

7.2.2 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).



7.2.3 ADC Operations

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

7.2.3.1 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.

7.2.3.2 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 $S(CG) \times N$ 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.

7.2.3.3 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)` `osalThreadResumeI(&(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 adclnit (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

Variables

- `ADCDriver ADCD1`
ADC1 driver identifier.

7.2.4 Macro Definition Documentation

7.2.4.1 `#define ADC_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.2.4.2 `#define ADC_USE_MUTUAL_EXCLUSION TRUE`

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

Note

Disabling this option saves both code and data space.

7.2.4.3 `#define _adc_reset_i(adcp) osalThreadResume1(&(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.

7.2.4.4 #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 ADCDriver object
----	-------------	---

Function Class:

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

7.2.4.5 #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 ADCDriver object
----	-------------	---

Function Class:

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

7.2.4.6 #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 ADCDriver object
----	-------------	---

Function Class:

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

7.2.4.7 #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 ADCDriver object
----	-------------	---

Function Class:

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

7.2.4.8 #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 ADCDriver object
----	-------------	---

Function Class:

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

7.2.4.9 #define _adc_isr_error_code(*adcp*, *err*)**Value:**

```

{
    adc_llc_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; \
    } \
    else { \
        (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 ADCDriver object
in	<i>err</i>	platform dependent error code

Function Class:

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

7.2.4.10 #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`.

7.2.5 Typedef Documentation

7.2.5.1 typedef uint16_t adcsample_t

ADC sample data type.

7.2.5.2 typedef uint16_t adc_channels_num_t

Channels number in a conversion group.

7.2.5.3 typedef struct ADCDriver ADCDriver

Type of a structure representing an ADC driver.

7.2.5.4 typedef void(* adccallback_t)(ADCDriver *adcp, adcsample_t *buffer, size_t n)

ADC notification callback type.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver 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>

7.2.5.5 typedef void(* adcerrorcallback_t)(ADCDriver *adcp, adcerror_t err)

ADC error callback type.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object triggering the callback
in	<i>err</i>	ADC error code

7.2.6 Enumeration Type Documentation

7.2.6.1 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 error.

7.2.6.2 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.

7.2.7 Function Documentation

7.2.7.1 void `adclnit` (void)

ADC Driver initialization.

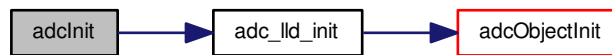
Note

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:



7.2.7.2 void `adcObjectInit` (`ADCDriver * adcp`)

Initializes the standard part of a `ADCDriver` structure.

Parameters

out	<i>adcp</i>	pointer to the <code>ADCDriver</code> 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:



7.2.7.3 void `adcStart` (`ADCDriver * adcp`, `const ADCCConfig * config`)

Configures and activates the ADC peripheral.

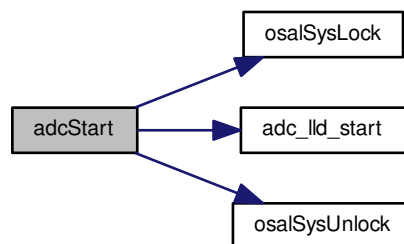
Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>config</i>	pointer to the ADCConfig object. Depending on the implementation the value can be <code>NULL</code> .

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:



7.2.7.4 void adcStop (ADCDriver * adcp)

Deactivates the ADC peripheral.

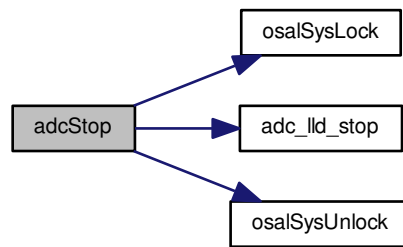
Parameters

in	<i>adcp</i>	pointer to the ADCDriver 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:



7.2.7.5 `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.

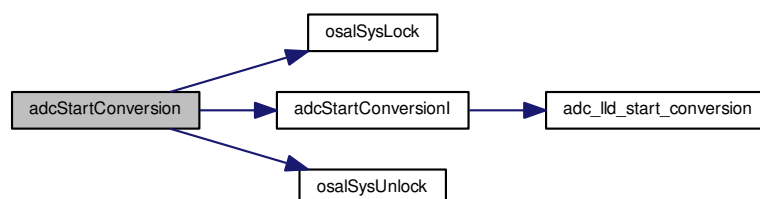
Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>grpp</i>	pointer to a ADCConversionGroup 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:



7.2.7.6 void `adcStartConversion1` (`ADCDriver * adcp`, const `ADCCConversionGroup * 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	<code>adcp</code>	pointer to the ADCDriver object
in	<code>grpp</code>	pointer to a ADCCConversionGroup 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.

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:



7.2.7.7 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.

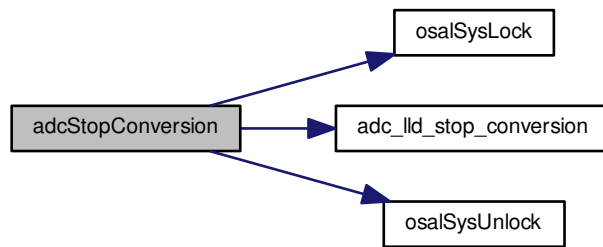
Parameters

in	<code>adcp</code>	pointer to the ADCDriver 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:



7.2.7.8 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 ADCDriver 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:



7.2.7.9 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	<i>adcp</i>	pointer to the ADCDriver object
in	<i>grpp</i>	pointer to a ADCConversionGroup 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.

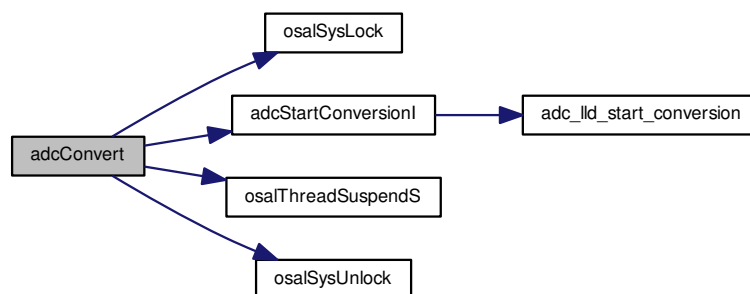
Return values

<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:

**7.2.7.10 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 ADCDriver 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:

**7.2.7.11 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 ADCDriver 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:



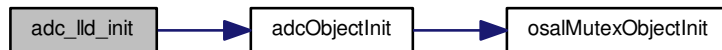
7.2.7.12 void adc_llc_init (void)

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:

**7.2.7.13 void adc_llc_start (ADCDriver * adcp)**

Configures and activates the ADC peripheral.

Parameters

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

Function Class:

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

7.2.7.14 void adc_llc_stop (ADCDriver * adcp)

Deactivates the ADC peripheral.

Parameters

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

Function Class:

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

7.2.7.15 void adc_llc_start_conversion (ADCDriver * adcp)

Starts an ADC conversion.

Parameters

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

Function Class:

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

7.2.7.16 void adc_ild_stop_conversion (ADCDriver * *adcp*)

Stops an ongoing conversion.

Parameters

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

Function Class:

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

7.2.8 Variable Documentation**7.2.8.1 ADCDriver ADCD1**

ADC1 driver identifier.

7.3 Generic Barometer Interface

HAL Generic Barometer Interface.

7.3.1 Detailed Description

HAL Generic Barometer Interface.

Macros

- `#define _base_barometer_methods_alone`
BaseBarometer specific methods.
- `#define _base_barometer_methods`
BaseBarometer specific methods with inherited ones.
- `#define _base_barometer_data _base_sensor_data`
BaseBarometer specific data.

Macro Functions (BaseBarometer)

- `#define barometerGetChannelsNumber(ip) (ip)->vmt->get_channels_number(ip)`
Barometer get channels number.
- `#define barometerReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)`
Barometer read raw data.
- `#define barometerReadCooked(ip, dp) (ip)->vmt->read_cooked(ip, dp)`
Barometer read cooked data.
- `#define barometerSetBias(ip, bp) (ip)->vmt->set_bias(ip, bp)`
Updates barometer bias data from received buffer.
- `#define barometerResetBias(ip) (ip)->vmt->reset_bias(ip)`
Reset barometer bias data restoring it to zero.
- `#define barometerSetSensitivity(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)`
Updates barometer sensitivity data from received buffer.
- `#define barometerResetSensitivity(ip) (ip)->vmt->reset_sensitivity(ip)`
Reset barometer sensitivity data restoring it to its typical value.

Data Structures

- `struct BaseBarometerVMT`
BaseBarometer virtual methods table.
- `struct BaseBarometer`
Base barometer class.

7.3.2 Macro Definition Documentation

7.3.2.1 `#define _base_barometer_methods_alone`

Value:

```

/* Invoke the set bias procedure.*/
msg_t (*set_bias)(void *instance, float biases[]);
/* Remove bias stored data.*/
msg_t (*reset_bias)(void *instance);
/* Invoke the set sensitivity procedure.*/
msg_t (*set_sensitivity)(void *instance, float sensitivities[]);
/* Restore sensitivity stored data to default.*/
msg_t (*reset_sensitivity)(void *instance);

```

[BaseBarometer](#) specific methods.

7.3.2.2 #define _base_barometer_methods

Value:

```

_base_sensor_methods
_base_barometer_methods_alone

```

[BaseBarometer](#) specific methods with inherited ones.

7.3.2.3 #define _base_barometer_data _base_sensor_data

[BaseBarometer](#) specific data.

7.3.2.4 #define barometerGetChannelsNumber(ip) (ip)->vmt->get_channels_number(ip)

Barometer get channels number.

Parameters

in	<i>ip</i>	pointer to a BaseBarometer class.
----	-----------	---

Returns

The number of channels of the [BaseBarometer](#)

Function Class:

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

7.3.2.5 #define barometerReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)

Barometer read raw data.

Parameters

in	<i>ip</i>	pointer to a BaseBarometer class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.3.2.6 `#define barometerReadCooked(ip, dp)(ip)->vmt->read_cooked(ip, dp)`

Barometer read cooked data.

Parameters

in	<i>ip</i>	pointer to a BaseBarometer class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.3.2.7 `#define barometerSetBias(ip, bp)(ip)->vmt->set_bias(ip, bp)`

Updates barometer bias data from received buffer.

Note

The bias buffer must have the same length of the the barometer channels number.

Parameters

in	<i>ip</i>	pointer to a BaseBarometer class.
in	<i>bp</i>	pointer to a buffer of bias values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.3.2.8 #define barometerResetBias(*ip*)(ip)->vmt->reset_bias(ip)

Reset barometer bias data restoring it to zero.

Parameters

in	<i>ip</i>	pointer to a BaseBarometer class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.3.2.9 #define barometerSetSensitivity(*ip, sp*)(ip)->vmt->set_sensitivity(ip, sp)

Updates barometer sensitivity data from received buffer.

Note

The sensitivity buffer must have the same length of the the barometer channels number.

Parameters

in	<i>ip</i>	pointer to a BaseBarometer class.
in	<i>sp</i>	pointer to a buffer of sensitivity values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.3.2.10 #define barometerResetSensitivity(ip)(ip)->vmt->reset_sensitivity(ip)

Reset barometer sensitivity data restoring it to its typical value.

Parameters

in	<i>ip</i>	pointer to a BaseBarometer class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.4 CAN Driver

Generic CAN Driver.

7.4.1 Detailed Description

Generic CAN Driver.

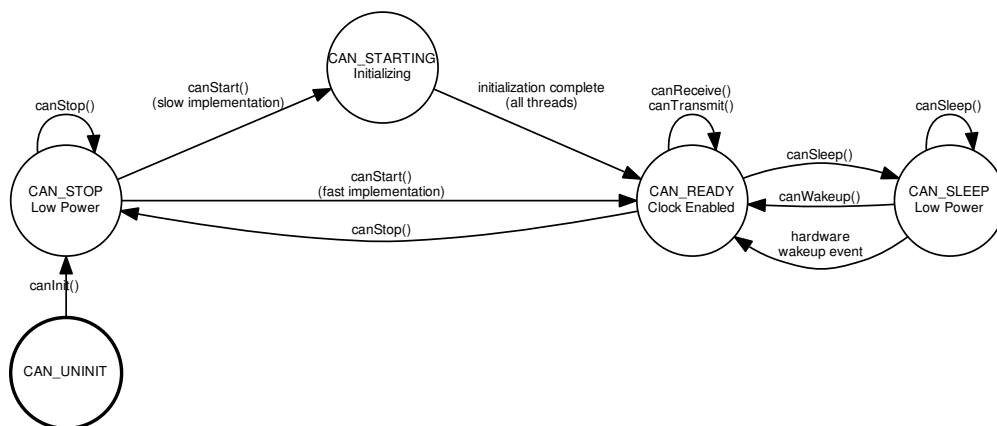
This module implements a generic CAN (Controller Area Network) driver allowing the exchange of information at frame level.

Precondition

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

7.4.2 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).



Macros

- `#define CAN_ANY_MAILBOX 0U`
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`

- *Errors rate warning.*
- #define `CAN_LIMIT_ERROR` 2U
- *Errors rate error.*
- #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.*
- #define `CAN_ENFORCE_USE_CALLBACKS` FALSE
- *Enforces the driver to use direct callbacks rather than OSAL events.*

Macro Functions

- #define `CAN_MAILBOX_TO_MASK(mbx)` (1U << ((mbx) - 1U))
- *Converts a mailbox index to a bit mask.*
- #define `canTransmit(canp, mailbox, ctfp, timeout)` `canTransmitTimeout(canp, mailbox, ctfp, timeout)`
- *Legacy name for `canTransmitTimeout()`.*
- #define `canReceive(canp, mailbox, crfp, timeout)` `canReceiveTimeout(canp, mailbox, crfp, timeout)`
- *Legacy name for `canReceiveTimeout()`.*

Low level driver helper macros

- #define `_can_tx_empty_isr(canp, flags)`
- *TX mailbox empty event.*
- #define `_can_rx_full_isr(canp, flags)`
- *RX mailbox empty full event.*
- #define `_can_wakeup_isr(canp)`
- *Error event.*
- #define `_can_error_isr(canp, flags)`
- *Error event.*

PLATFORM configuration options

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

Typedefs

- typedef struct `CANDriver` `CANDriver`
- *Type of a structure representing an CAN driver.*
- typedef uint32_t `canmbx_t`
- *Type of a transmission mailbox index.*
- typedef void(* `can_callback_t`) (`CANDriver` *canp, uint32_t flags)
- *Type of a CAN notification callback.*

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 *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 [canTryTransmitI](#) (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)
Can frame transmission attempt.
- bool [canTryReceiveI](#) (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
Can frame receive attempt.
- msg_t [canTransmitTimeout](#) (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, sysinterval_t timeout)
Can frame transmission.
- msg_t [canReceiveTimeout](#) (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, sysinterval_t timeout)
Can frame receive.
- void [canSleep](#) (CANDriver *canp)
Enters the sleep mode.
- void [canWakeup](#) (CANDriver *canp)
Enforces leaving the sleep mode.
- void [can_llc_init](#) (void)
Low level CAN driver initialization.
- void [can_llc_start](#) (CANDriver *canp)
Configures and activates the CAN peripheral.
- void [can_llc_stop](#) (CANDriver *canp)
Deactivates the CAN peripheral.
- bool [can_llc_is_tx_empty](#) (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame can be transmitted.
- void [can_llc_transmit](#) (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)
Inserts a frame into the transmit queue.
- bool [can_llc_is_rx_nonempty](#) (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame has been received.
- void [can_llc_receive](#) (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
Receives a frame from the input queue.
- void [can_llc_sleep](#) (CANDriver *canp)
Enters the sleep mode.
- void [can_llc_wakeup](#) (CANDriver *canp)
Enforces leaving the sleep mode.

Enumerations

Variables

- [CANDriver CAND1](#)

CAN1 driver identifier.

7.4.3 Macro Definition Documentation

7.4.3.1 #define CAN_LIMIT_WARNING 1U

Errors rate warning.

7.4.3.2 #define CAN_LIMIT_ERROR 2U

Errors rate error.

7.4.3.3 #define CAN_BUS_OFF_ERROR 4U

Bus off condition reached.

7.4.3.4 #define CAN_FRAMING_ERROR 8U

Framing error of some kind on the CAN bus.

7.4.3.5 #define CAN_OVERFLOW_ERROR 16U

Overflow in receive queue.

7.4.3.6 #define CAN_ANY_MAILBOX 0U

Special mailbox identifier.

7.4.3.7 #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.

7.4.3.8 #define CAN_ENFORCE_USE_CALLBACKS FALSE

Enforces the driver to use direct callbacks rather than OSAL events.

7.4.3.9 #define CAN_MAILBOX_TO_MASK(mbx) (1U << ((mbx) - 1U))

Converts a mailbox index to a bit mask.

7.4.3.10 `#define canTransmit(canp, mailbox, ctfp, timeout) canTransmitTimeout(canp, mailbox, ctfp, timeout)`

Legacy name for `canTransmitTimeout()`.

Deprecated

7.4.3.11 `#define canReceive(canp, mailbox, crfp, timeout) canReceiveTimeout(canp, mailbox, crfp, timeout)`

Legacy name for `canReceiveTimeout()`.

Deprecated

7.4.3.12 `#define _can_tx_empty_isr(canp, flags)`

Value:

```
{
    osalSysLockFromISR();
    osalThreadDequeueAllI (&(canp)->txqueue, MSG_OK);
    osalEventBroadcastFlagsI (&(canp)->txempty_event, flags);
    osalSysUnlockFromISR();
}
```

TX mailbox empty event.

7.4.3.13 `#define _can_rx_full_isr(canp, flags)`

Value:

```
{
    osalSysLockFromISR();
    osalThreadDequeueAllI (&(canp)->rxqueue, MSG_OK);
    osalEventBroadcastFlagsI (&(canp)->rxfull_event, flags);
    osalSysUnlockFromISR();
}
```

RX mailbox empty full event.

7.4.3.14 `#define _can_wakeup_isr(canp)`

Value:

```
{
    osalSysLockFromISR();
    osalEventBroadcastFlagsI (&(canp)->wakeup_event, 0U);
    osalSysUnlockFromISR();
}
```

Error event.

7.4.3.15 `#define _can_error_isr(canp, flags)`**Value:**

```

{
    osalSysLockFromISR();
    osalEventBroadcastFlagsI(&(canp)->error_event, flags);
    osalSysUnlockFromISR();
}

```

Error event.

7.4.3.16 `#define CAN_TX_MAILBOXES 1`

Number of transmit mailboxes.

7.4.3.17 `#define CAN_RX_MAILBOXES 1`

Number of receive mailboxes.

7.4.3.18 `#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`.**7.4.4 Typedef Documentation**7.4.4.1 `typedef struct CANDriver CANDriver`

Type of a structure representing an CAN driver.

7.4.4.2 `typedef uint32_t canmbx_t`

Type of a transmission mailbox index.

7.4.4.3 `typedef void(* can_callback_t) (CANDriver *canp, uint32_t flags)`

Type of a CAN notification callback.

Parameters

in	<i>canp</i>	pointer to the CANDriver object triggering the callback
in	<i>flags</i>	flags associated to the mailbox callback

7.4.5 Enumeration Type Documentation

7.4.5.1 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.

7.4.6 Function Documentation

7.4.6.1 void canInit (void)

CAN Driver initialization.

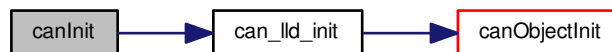
Note

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:



7.4.6.2 void canObjectInit (CANDriver * canp)

Initializes the standard part of a `CANDriver` structure.

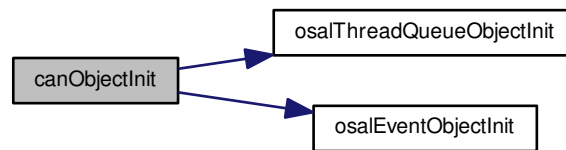
Parameters

out	<i>canp</i>	pointer to the <code>CANDriver</code> 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:



7.4.6.3 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\(\)](#).

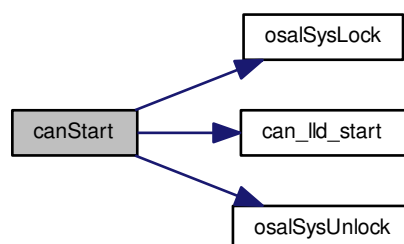
Parameters

in	<i>canp</i>	pointer to the CANDriver object
in	<i>config</i>	pointer to the CANConfig object. Depending on the implementation the value can be <code>NULL</code> .

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:



7.4.6.4 void canStop (CANDriver * canp)

Deactivates the CAN peripheral.

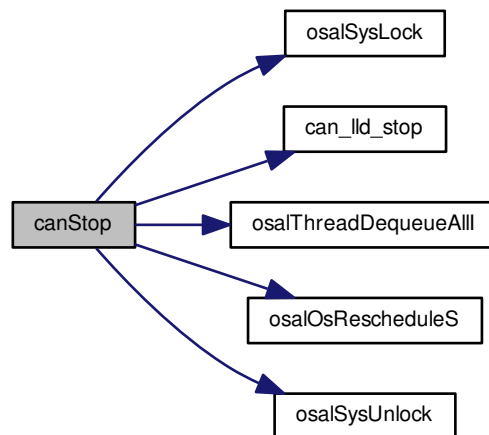
Parameters

in	<i>canp</i>	pointer to the CANDriver 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:



7.4.6.5 `bool canTryTransmitl (CANDriver * canp, canmbx_t mailbox, const CANTxFrame * 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 CANDriver 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

Returns

The operation result.

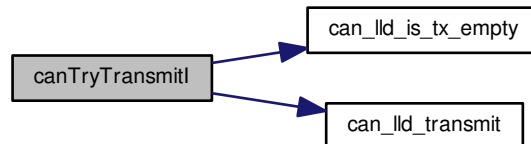
Return values

<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:



7.4.6.6 bool canTryReceiveI (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 CANDriver 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.

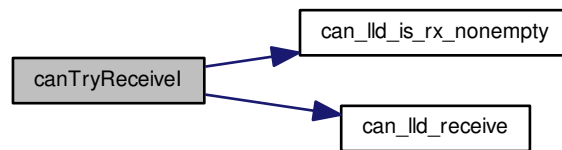
Return values

<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:



7.4.6.7 `msg_t canTransmitTimeout (CANDriver * canp, canmbx_t mailbox, const CANTxFrame * ctfp, sysinterval_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 CANDriver 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
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation result.

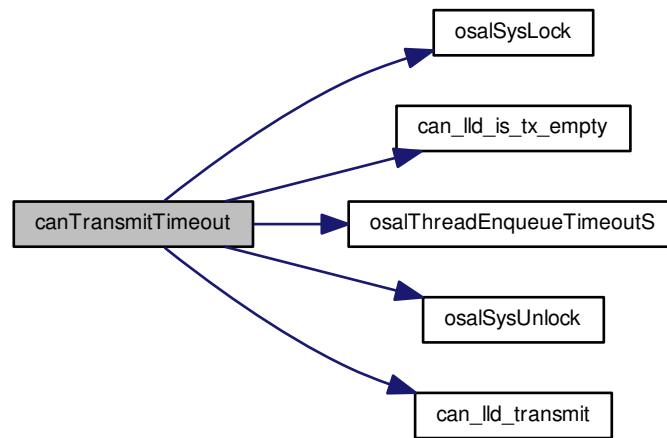
Return values

<i>MSG_OK</i>	the frame has been queued for transmission.
<i>MSG_TIMEOUT</i>	The operation has timed out.
<i>MSG_RESET</i>	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:



7.4.6.8 `msg_t canReceiveTimeout (CANDriver * canp, canmbx_t mailbox, CANRxFrame * crfp, sysinterval_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	<i>canp</i>	pointer to the CANDriver 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
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout (useful in an event driven scenario where a thread never blocks for I/O). • <code>TIME_INFINITE</code> no timeout.

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.

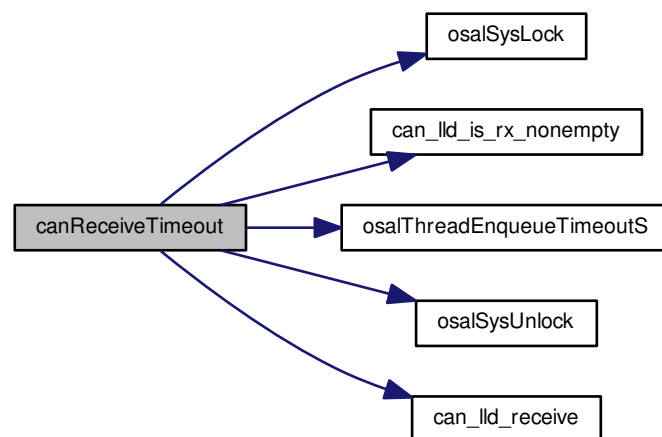
Return values

<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:



7.4.6.9 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.

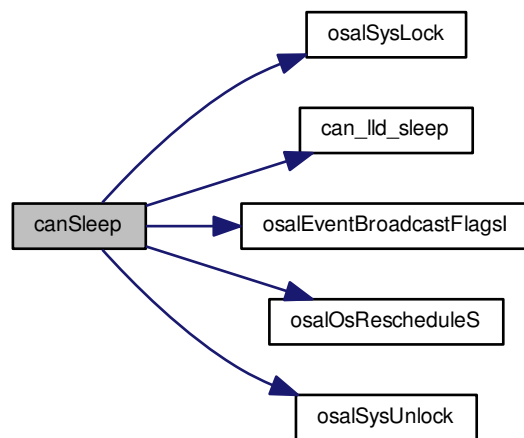
Parameters

in	<code>canp</code>	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:



7.4.6.10 void canWakeup (CANDriver * *canp*)

Enforces leaving the sleep mode.

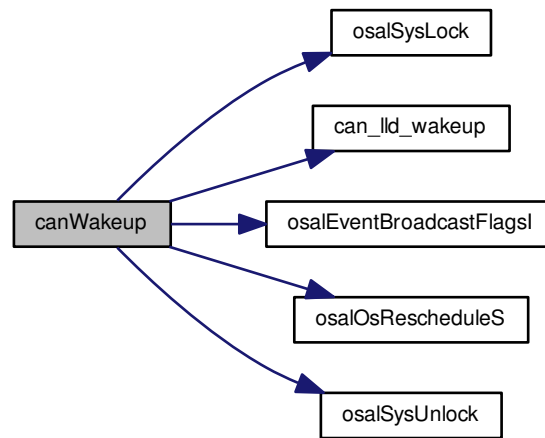
Note

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

Parameters

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

Here is the call graph for this function:



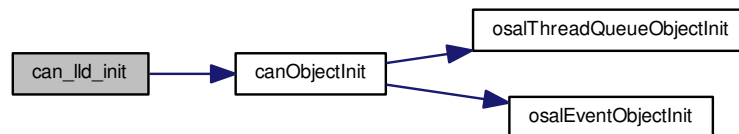
7.4.6.11 void can_lld_init (void)

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:



7.4.6.12 void can_lld_start (CANDriver * canp)

Configures and activates the CAN peripheral.

Parameters

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

Function Class:

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

7.4.6.13 void can_ild_stop (CANDriver * *canp*)

Deactivates the CAN peripheral.

Parameters

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

Function Class:

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

7.4.6.14 bool can_ild_is_tx_empty (CANDriver * *canp*, canmbx_t *mailbox*)

Determines whether a frame can be transmitted.

Parameters

in	<i>canp</i>	pointer to the CANDriver 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.

7.4.6.15 void can_ild_transmit (CANDriver * *canp*, canmbx_t *mailbox*, const CANTxFrame * *ctfp*)

Inserts a frame into the transmit queue.

Parameters

in	<i>canp</i>	pointer to the CANDriver 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.

7.4.6.16 bool can_ild_is_rx_nonempty (CANDriver * *canp*, canmbx_t *mailbox*)

Determines whether a frame has been received.

Parameters

in	<i>canp</i>	pointer to the CANDriver 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.

7.4.6.17 void can_ild_receive (CANDriver * *canp*, canmbx_t *mailbox*, CANRxFrame * *crfp*)

Receives a frame from the input queue.

Parameters

in	<i>canp</i>	pointer to the CANDriver 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.

7.4.6.18 void can_ild_sleep (CANDriver * *canp*)

Enters the sleep mode.

Parameters

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

Function Class:

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

7.4.6.19 void can_ild_wakeup (CANDriver * *canp*)

Enforces leaving the sleep mode.

Parameters

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

Function Class:

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

7.4.7 Variable Documentation**7.4.7.1 CANDriver CAND1**

CAN1 driver identifier.

7.5 Generic Compass Interface

HAL Generic Compass Interface.

7.5.1 Detailed Description

HAL Generic Compass Interface.

Macros

- `#define _base_compass_methods_alone`
BaseCompass specific methods.
- `#define _base_compass_methods`
BaseCompass specific methods with inherited ones.
- `#define _base_compass_data _base_sensor_data`
BaseCompass specific data.

Macro Functions (BaseCompass)

- `#define compassGetAxesNumber(ip) (ip)->vmt->get_channels_number(ip)`
Compass get axes number.
- `#define compassReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)`
Compass read raw data.
- `#define compassReadCooked(ip, dp) (ip)->vmt->read_cooked(ip, dp)`
Compass read cooked data.
- `#define compassSetBias(ip, bp) (ip)->vmt->set_bias(ip, bp)`
Updates compass bias data from received buffer.
- `#define compassResetBias(ip) (ip)->vmt->reset_bias(ip)`
Reset compass bias data restoring it to zero.
- `#define compassSetSensitivity(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)`
Updates compass sensitivity data from received buffer.
- `#define compassResetSensitivity(ip) (ip)->vmt->reset_sensitivity(ip)`
Reset compass sensitivity data restoring it to its typical value.

Data Structures

- struct `BaseCompassVMT`
BaseCompass virtual methods table.
- struct `BaseCompass`
Base compass class.

7.5.2 Macro Definition Documentation

7.5.2.1 `#define _base_compass_methods_alone`

Value:

```

/* Invoke the set bias procedure.*/
msg_t (*set_bias)(void *instance, float biases[]);
/* Remove bias stored data.*/
msg_t (*reset_bias)(void *instance);
/* Invoke the set sensitivity procedure.*/
msg_t (*set_sensitivity)(void *instance, float sensitivities[]);
/* Restore sensitivity stored data to default.*/
msg_t (*reset_sensitivity)(void *instance);

```

[BaseCompass](#) specific methods.

7.5.2.2 #define _base_compass_methods

Value:

```

_base_sensor_methods
_base_compass_methods_alone

```

[BaseCompass](#) specific methods with inherited ones.

7.5.2.3 #define _base_compass_data _base_sensor_data

[BaseCompass](#) specific data.

7.5.2.4 #define compassGetAxesNumber(ip) (ip)->vmt->get_channels_number(ip)

Compass get axes number.

Parameters

in	<i>ip</i>	pointer to a BaseCompass class.
----	-----------	---

Returns

The number of axes of the [BaseCompass](#)

Function Class:

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

7.5.2.5 #define compassReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)

Compass read raw data.

Parameters

in	<i>ip</i>	pointer to a BaseCompass class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.5.2.6 #define compassReadCooked(ip, dp)(ip)->vmt->read_cooked(ip, dp)

Compass read cooked data.

Parameters

in	<i>ip</i>	pointer to a BaseCompass class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.5.2.7 #define compassSetBias(ip, bp)(ip)->vmt->set_bias(ip, bp)

Updates compass bias data from received buffer.

Note

The bias buffer must have the same length of the the compass axes number.

Parameters

in	<i>ip</i>	pointer to a BaseCompass class.
in	<i>bp</i>	pointer to a buffer of bias values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.5.2.8 #define compassResetBias(*ip*)(ip)->vmt->reset_bias(ip)

Reset compass bias data restoring it to zero.

Parameters

in	<i>ip</i>	pointer to a BaseCompass class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.5.2.9 #define compassSetSensitivity(*ip, sp*)(ip)->vmt->set_sensitivity(ip, sp)

Updates compass sensitivity data from received buffer.

Note

The sensitivity buffer must have the same length of the the compass axes number.

Parameters

in	<i>ip</i>	pointer to a BaseCompass class.
in	<i>sp</i>	pointer to a buffer of sensitivity values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.5.2.10 #define compassResetSensitivity(*ip*) (ip)->vmt->reset_sensitivity(ip)

Reset compass sensitivity data restoring it to its typical value.

Parameters

in	<i>ip</i>	pointer to a BaseCompass class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.6 Crypto Driver

Generic Crypto Driver.

7.6.1 Detailed Description

Generic Crypto Driver.

This module implements a generic Cryptography driver.

Precondition

In order to use the I2C driver the `HAL_USE_CRY` option must be enabled in `halconf.h`.

Macros

- `#define HAL_CRY_MAX_KEY_SIZE 32`
Maximum size of a key for all supported algorithms.
- `#define HAL_CRY_USE_FALLBACK FALSE`
Enables the SW fall-back of the cryptographic driver.
- `#define HAL_CRY_USE_FALLBACK TRUE`
Enables the SW fall-back of the cryptographic driver.
- `#define HAL_CRY_ENFORCE_FALLBACK FALSE`
Makes the driver forcibly use the fall-back implementations.

Driver capability switches

- `#define CRY_LLD_SUPPORTS_AES TRUE`
- `#define CRY_LLD_SUPPORTS_AES_ECB TRUE`
- `#define CRY_LLD_SUPPORTS_AES_CBC TRUE`
- `#define CRY_LLD_SUPPORTS_AES_CFB TRUE`
- `#define CRY_LLD_SUPPORTS_AES_CTR TRUE`
- `#define CRY_LLD_SUPPORTS_AES_GCM TRUE`
- `#define CRY_LLD_SUPPORTS_DES TRUE`
- `#define CRY_LLD_SUPPORTS_DES_ECB TRUE`
- `#define CRY_LLD_SUPPORTS_DES_CBC TRUE`
- `#define CRY_LLD_SUPPORTS_SHA1 TRUE`
- `#define CRY_LLD_SUPPORTS_SHA256 TRUE`
- `#define CRY_LLD_SUPPORTS_SHA512 TRUE`
- `#define CRY_LLD_SUPPORTS_TRNG TRUE`

PLATFORM configuration options

- `#define PLATFORM_CRY_USE_CRY1 FALSE`
CRY1 driver enable switch.

Typedefs

- `typedef size_t bitsize_t`
Size, in bits, of a crypto field or message.
- `typedef uint32_t crykey_t`
CRY key identifier type.
- `typedef struct CRYDriver CRYDriver`
Type of a structure representing an CRY driver.

Data Structures

- struct [SHA1Context](#)
Type of a SHA1 context.
- struct [SHA256Context](#)
Type of a SHA256 context.
- struct [SHA512Context](#)
Type of a SHA512 context.
- struct [CRYConfig](#)
Driver configuration structure.
- struct [CRYDriver](#)
Structure representing an CRY driver.

Functions

- void [cryInit](#) (void)
Cryptographic Driver initialization.
- void [cryObjectInit](#) (CRYDriver *cryp)
Initializes the standard part of a CRYDriver structure.
- void [cryStart](#) (CRYDriver *cryp, const CRYConfig *config)
Configures and activates the cryptographic peripheral.
- void [cryStop](#) (CRYDriver *cryp)
Deactivates the cryptographic peripheral.
- [cryerror_t](#) [cryLoadTransientKey](#) (CRYDriver *cryp, [cryalgorithm_t](#) algorithm, size_t size, const uint8_t *keyp)
Initializes the transient key for a specific algorithm.
- [cryerror_t](#) [cryEncryptAES](#) (CRYDriver *cryp, [crykey_t](#) key_id, const uint8_t *in, uint8_t *out)
Encryption of a single block using AES.
- [cryerror_t](#) [cryDecryptAES](#) (CRYDriver *cryp, [crykey_t](#) key_id, const uint8_t *in, uint8_t *out)
Decryption of a single block using AES.
- [cryerror_t](#) [cryEncryptAES_ECB](#) (CRYDriver *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out)
Encryption operation using AES-ECB.
- [cryerror_t](#) [cryDecryptAES_ECB](#) (CRYDriver *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out)
Decryption operation using AES-ECB.
- [cryerror_t](#) [cryEncryptAES_CBC](#) (CRYDriver *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CBC.
- [cryerror_t](#) [cryDecryptAES_CBC](#) (CRYDriver *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Decryption operation using AES-CBC.
- [cryerror_t](#) [cryEncryptAES_CFB](#) (CRYDriver *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CFB.
- [cryerror_t](#) [cryDecryptAES_CFB](#) (CRYDriver *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Decryption operation using AES-CFB.
- [cryerror_t](#) [cryEncryptAES_CTR](#) (CRYDriver *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CTR.
- [cryerror_t](#) [cryDecryptAES_CTR](#) (CRYDriver *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Decryption operation using AES-CTR.

- `cryerror_t cryEncryptAES_GCM` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`, `size_t aadsize`, `const uint8_t *aad`, `uint8_t *authtag`)

Encryption operation using AES-GCM.

- `cryerror_t cryDecryptAES_GCM` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`, `size_t aadsize`, `const uint8_t *aad`, `uint8_t *authtag`)

Decryption operation using AES-GCM.

- `cryerror_t cryEncryptDES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)

Encryption of a single block using (T)DES.

- `cryerror_t cryDecryptDES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)

Decryption of a single block using (T)DES.

- `cryerror_t cryEncryptDES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)

Encryption operation using (T)DES-ECB.

- `cryerror_t cryDecryptDES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)

Decryption operation using (T)DES-ECB.

- `cryerror_t cryEncryptDES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Encryption operation using (T)DES-CBC.

- `cryerror_t cryDecryptDES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Decryption operation using (T)DES-CBC.

- `cryerror_t crySHA1Init` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`)

Hash initialization using SHA1.

- `cryerror_t crySHA1Update` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `size_t size`, `const uint8_t *in`)

Hash update using SHA1.

- `cryerror_t crySHA1Final` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `uint8_t *out`)

Hash finalization using SHA1.

- `cryerror_t crySHA256Init` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`)

Hash initialization using SHA256.

- `cryerror_t crySHA256Update` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `size_t size`, `const uint8_t *in`)

Hash update using SHA256.

- `cryerror_t crySHA256Final` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `uint8_t *out`)

Hash finalization using SHA256.

- `cryerror_t crySHA512Init` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`)

Hash initialization using SHA512.

- `cryerror_t crySHA512Update` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `size_t size`, `const uint8_t *in`)

Hash update using SHA512.

- `cryerror_t crySHA512Final` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `uint8_t *out`)

Hash finalization using SHA512.

- `cryerror_t cryTRNG` (`CRYDriver *cryp`, `uint8_t *out`)

True random numbers generator.

- `void cry_lld_init` (`void`)

Low level crypto driver initialization.

- `void cry_lld_start` (`CRYDriver *cryp`)

Configures and activates the crypto peripheral.

- `void cry_lld_stop` (`CRYDriver *cryp`)

Deactivates the crypto peripheral.

- `cryerror_t cry_lld_loadkey` (`CRYDriver *cryp`, `cryalgorithm_t algorithm`, `size_t size`, `const uint8_t *keyp`)

Initializes the transient key for a specific algorithm.

- `cryerror_t cry_lld_encrypt_AES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)
Encryption of a single block using AES.
- `cryerror_t cry_lld_decrypt_AES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)
Decryption of a single block using AES.
- `cryerror_t cry_lld_encrypt_AES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)
Encryption operation using AES-ECB.
- `cryerror_t cry_lld_decrypt_AES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)
Decryption operation using AES-ECB.
- `cryerror_t cry_lld_encrypt_AES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Encryption operation using AES-CBC.
- `cryerror_t cry_lld_decrypt_AES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Decryption operation using AES-CBC.
- `cryerror_t cry_lld_encrypt_AES_CFB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Encryption operation using AES-CFB.
- `cryerror_t cry_lld_decrypt_AES_CFB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Decryption operation using AES-CFB.
- `cryerror_t cry_lld_encrypt_AES_CTR` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Encryption operation using AES-CTR.
- `cryerror_t cry_lld_decrypt_AES_CTR` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Decryption operation using AES-CTR.
- `cryerror_t cry_lld_encrypt_AES_GCM` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`, `size_t aadsize`, `const uint8_t *aad`, `uint8_t *authtag`)
Encryption operation using AES-GCM.
- `cryerror_t cry_lld_decrypt_AES_GCM` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`, `size_t aadsize`, `const uint8_t *aad`, `uint8_t *authtag`)
Decryption operation using AES-GCM.
- `cryerror_t cry_lld_encrypt_DES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)
Encryption of a single block using (T)DES.
- `cryerror_t cry_lld_decrypt_DES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)
Decryption of a single block using (T)DES.
- `cryerror_t cry_lld_encrypt_DES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)
Encryption operation using (T)DES-ECB.
- `cryerror_t cry_lld_decrypt_DES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)
Decryption operation using (T)DES-ECB.
- `cryerror_t cry_lld_encrypt_DES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Encryption operation using (T)DES-CBC.
- `cryerror_t cry_lld_decrypt_DES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Decryption operation using (T)DES-CBC.
- `cryerror_t cry_lld_SHA1_init` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`)
Hash initialization using SHA1.

- `cryerror_t cry_lld_SHA1_update` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `size_t size`, `const uint8_t *in`)
Hash update using SHA1.
- `cryerror_t cry_lld_SHA1_final` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `uint8_t *out`)
Hash finalization using SHA1.
- `cryerror_t cry_lld_SHA256_init` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`)
Hash initialization using SHA256.
- `cryerror_t cry_lld_SHA256_update` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `size_t size`, `const uint8_t *in`)
Hash update using SHA256.
- `cryerror_t cry_lld_SHA256_final` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `uint8_t *out`)
Hash finalization using SHA256.
- `cryerror_t cry_lld_SHA512_init` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`)
Hash initialization using SHA512.
- `cryerror_t cry_lld_SHA512_update` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `size_t size`, `const uint8_t *in`)
Hash update using SHA512.
- `cryerror_t cry_lld_SHA512_final` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `uint8_t *out`)
Hash finalization using SHA512.
- `cryerror_t cry_lld_TRNG` (`CRYDriver *cryp`, `uint8_t *out`)
True random numbers generator.

Enumerations

Variables

- `CRYDriver CRYD1`
CRY1 driver identifier.

7.6.2 Macro Definition Documentation

7.6.2.1 `#define HAL_CRY_MAX_KEY_SIZE 32`

Maximum size of a key for all supported algorithms.

7.6.2.2 `#define HAL_CRY_USE_FALLBACK FALSE`

Enables the SW fall-back of the cryptographic driver.

When enabled, this option, activates a fall-back software implementation for algorithms not supported by the underlying hardware.

Note

Fall-back implementations may not be present for all algorithms.

7.6.2.3 `#define HAL_CRY_USE_FALLBACK TRUE`

Enables the SW fall-back of the cryptographic driver.

When enabled, this option, activates a fall-back software implementation for algorithms not supported by the underlying hardware.

Note

Fall-back implementations may not be present for all algorithms.

7.6.2.4 `#define HAL_CRY_ENFORCE_FALLBACK FALSE`

Makes the driver forcibly use the fall-back implementations.

Note

If enabled then the LLD driver is not included at all.

7.6.2.5 `#define PLATFORM_CRY_USE_CRY1 FALSE`

CRY1 driver enable switch.

If set to `TRUE` the support for CRY1 is included.

Note

The default is `FALSE`.

7.6.3 Typedef Documentation

7.6.3.1 `typedef size_t bitsize_t`

Size, in bits, of a crypto field or message.

Note

It is assumed, for simplicity, that this type is equivalent to a `size_t`.

7.6.3.2 `typedef uint32_t crykey_t`

CRY key identifier type.

7.6.3.3 `typedef struct CRYDriver CRYDriver`

Type of a structure representing an CRY driver.

7.6.4 Enumeration Type Documentation

7.6.4.1 `enum crystate_t`

Driver state machine possible states.

Enumerator

`CRY_UNINIT` Not initialized.

`CRY_STOP` Stopped.

`CRY_READY` Ready.

7.6.4.2 enum cryerror_t

Driver error codes.

Enumerator

CRY_NOERROR No error.
CRY_ERR_INV_ALGO Invalid cypher/mode.
CRY_ERR_INV_KEY_SIZE Invalid key size.
CRY_ERR_INV_KEY_TYPE Invalid key type.
CRY_ERR_INV_KEY_ID Invalid key type.

7.6.4.3 enum cryalgorithm_t

Type of an algorithm identifier.

Note

It is only used to determine the key required for operations.

Enumerator

cry_algo_aes AES 128, 192, 256 bits.
cry_algo_des DES 56, TDES 112, 168 bits.

7.6.5 Function Documentation

7.6.5.1 void cryInit (void)

Cryptographic 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:



7.6.5.2 void cryObjectInit (CRYDriver * cryp)

Initializes the standard part of a `CRYDriver` structure.

Parameters

out	<i>cryp</i>	pointer to the CRYDriver object
-----	-------------	---

Function Class:

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

7.6.5.3 void cryStart (CRYDriver * *cryp*, const CRYConfig * *config*)

Configures and activates the cryptographic peripheral.

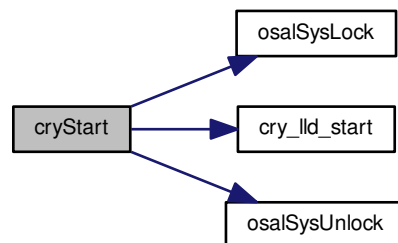
Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>config</i>	pointer to the CRYConfig 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:

**7.6.5.4 void cryStop (CRYDriver * *cryp*)**

Deactivates the cryptographic peripheral.

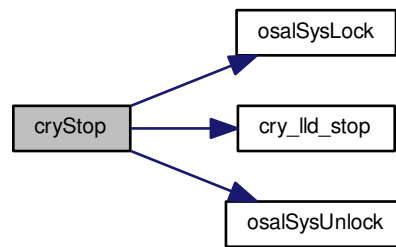
Parameters

in	<i>cryp</i>	pointer to the CRYDriver 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:



7.6.5.5 `cryerror_t cryLoadTransientKey (CRYDriver * cryp, cryalgorithm_t algorithm, size_t size, const uint8_t * keyp)`

Initializes the transient key for a specific algorithm.

Note

It is the underlying implementation to decide which combinations of algorithm and key size are allowable.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>algorithm</i>	the algorithm identifier
in	<i>size</i>	key size in bytes
in	<i>keyp</i>	pointer to the key data

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the specified algorithm is unknown or unsupported.
<code>CRY_ERR_INV_KEY_SIZE</code>	if the specified key size is invalid for the specified algorithm.

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:



7.6.5.6 `cryerror_t cryEncryptAES (CRYDriver * cryp, crykey_t key_id, const uint8_t * in, uint8_t * out)`

Encryption of a single block using AES.

Note

The implementation of this function must guarantee that it can be called from any context.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

Here is the call graph for this function:



7.6.5.7 `cryerror_t cryDecryptAES (CRYDriver * cryp, crykey_t key_id, const uint8_t * in, uint8_t * out)`

Decryption of a single block using AES.

Note

The implementation of this function must guarantee that it can be called from any context.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

Here is the call graph for this function:



7.6.5.8 `cryerror_t cryEncryptAES_ECB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out)`

Encryption operation using AES-ECB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.9 `cryerror_t cryDecryptAES_ECB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out)`

Decryption operation using AES-ECB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.10 `cryerror_t cryEncryptAES_CBC (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Encryption operation using AES-CBC.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits input vector

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.11 `cryerror_t cryDecryptAES_CBC (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Decryption operation using AES-CBC.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext
in	<i>iv</i>	128 bits input vector

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.12 `cryerror_t cryEncryptAES_CFB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Encryption operation using AES-CFB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits input vector

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.13 `cryerror_t cryDecryptAES_CFB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Decryption operation using AES-CFB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext
in	<i>iv</i>	128 bits input vector

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.14 `cryerror_t cryEncryptAES_CTR (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Encryption operation using AES-CTR.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits input vector + counter, it contains a 96 bits IV and a 32 bits counter

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.15 `cryerror_t cryDecryptAES_CTR (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Decryption operation using AES-CTR.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext
in	<i>iv</i>	128 bits input vector + counter, it contains a 96 bits IV and a 32 bits counter

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.16 `cryerror_t cryEncryptAES_GCM (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv, size_t aadsize, const uint8_t * aad, uint8_t * authtag)`

Encryption operation using AES-GCM.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the text buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits input vector + counter, it contains a 96 bits IV and a 32 bits counter
in	<i>aadsize</i>	size of the authentication data, this number must be a multiple of 16
in	<i>aad</i>	buffer containing the authentication data
in	<i>authtag</i>	128 bits buffer for the generated authentication tag

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.17 `cryerror_t cryDecryptAES_GCM (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv, size_t aadsize, const uint8_t * aad, uint8_t * authtag)`

Decryption operation using AES-GCM.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the text buffers, this number must be a multiple of 16
in	<i>in</i>	buffer for the output cyphertext
out	<i>out</i>	buffer containing the input plaintext
in	<i>iv</i>	128 bits input vector + counter, it contains a 96 bits IV and a 32 bits counter
in	<i>aadsize</i>	size of the authentication data, this number must be a multiple of 16
in	<i>aad</i>	buffer containing the authentication data
in	<i>authtag</i>	128 bits buffer for the generated authentication tag

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.18 `cryerror_t cryEncryptDES (CRYDriver * cryp, crykey_t key_id, const uint8_t * in, uint8_t * out)`

Encryption of a single block using (T)DES.

Note

The implementation of this function must guarantee that it can be called from any context.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

Here is the call graph for this function:



7.6.5.19 `cryerror_t cryDecryptDES (CRYDriver * cryp, crykey_t key_id, const uint8_t * in, uint8_t * out)`

Decryption of a single block using (T)DES.

Note

The implementation of this function must guarantee that it can be called from any context.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

Here is the call graph for this function:



7.6.5.20 `cryerror_t cryEncryptDES_ECB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out)`

Encryption operation using (T)DES-ECB.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 8
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.21 `cryerror_t cryDecryptDES_ECB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out)`

Decryption operation using (T)DES-ECB.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 8
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.22 `cryerror_t cryEncryptDES_CBC (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Encryption operation using (T)DES-CBC.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 8
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	64 bits input vector

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.23 `cryerror_t cryDecryptDES_CBC (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Decryption operation using (T)DES-CBC.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of both buffers, this number must be a multiple of 8
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext
in	<i>iv</i>	64 bits input vector

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

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:



7.6.5.24 `cryerror_t crySHA1Init (CRYDriver * crypt, SHA1Context * sha1ctxp)`

Hash initialization using SHA1.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>crypt</i>	pointer to the CRYDriver object
out	<i>sha1ctxp</i>	pointer to a SHA1 context to be initialized

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.

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:



7.6.5.25 `cryerror_t crySHA1Update (CRYDriver * cryp, SHA1Context * sha1ctxp, size_t size, const uint8_t * in)`

Hash update using SHA1.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha1ctxp</i>	pointer to a SHA1 context
in	<i>size</i>	size of input buffer
in	<i>in</i>	buffer containing the input text

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

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:



7.6.5.26 `cryerror_t crySHA1Final (CRYDriver * cryp, SHA1Context * sha1ctxp, uint8_t * out)`

Hash finalization using SHA1.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha1ctxp</i>	pointer to a SHA1 context
out	<i>out</i>	160 bits output buffer

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

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:



7.6.5.27 `cryerror_t crySHA256Init (CRYDriver * cryp, SHA256Context * sha256ctxp)`

Hash initialization using SHA256.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
out	<i>sha256ctxp</i>	pointer to a SHA256 context to be initialized

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

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:



7.6.5.28 `cryerror_t crySHA256Update (CRYDriver * cryp, SHA256Context * sha256ctxp, size_t size, const uint8_t * in)`

Hash update using SHA256.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha256ctxp</i>	pointer to a SHA256 context
in	<i>size</i>	size of input buffer
in	<i>in</i>	buffer containing the input text

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.

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:



7.6.5.29 `cryerror_t crySHA256Final (CRYDriver * cryp, SHA256Context * sha256ctxp, uint8_t * out)`

Hash finalization using SHA256.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha256ctxp</i>	pointer to a SHA256 context
out	<i>out</i>	256 bits output buffer

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

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:



7.6.5.30 `cryerror_t crySHA512Init (CRYDriver * cryp, SHA512Context * sha512ctxp)`

Hash initialization using SHA512.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
out	<i>sha512ctxp</i>	pointer to a SHA512 context to be initialized

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

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:



7.6.5.31 `cryerror_t crySHA512Update (CRYDriver * cryp, SHA512Context * sha512ctxp, size_t size, const uint8_t * in)`

Hash update using SHA512.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha512ctxp</i>	pointer to a SHA512 context
in	<i>size</i>	size of input buffer
in	<i>in</i>	buffer containing the input text

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

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:



7.6.5.32 `cryerror_t crySHA512Final (CRYDriver * cryp, SHA512Context * sha512ctxp, uint8_t * out)`

Hash finalization using SHA512.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha512ctxp</i>	pointer to a SHA512 context
out	<i>out</i>	512 bits output buffer

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.

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:



7.6.5.33 `cryerror_t cryTRNG (CRYDriver * cryp, uint8_t * out)`

True random numbers generator.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
out	<i>out</i>	128 bits output buffer

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.

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:

**7.6.5.34** `void cry_lld_init (void)`

Low level crypto driver initialization.

Function Class:

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

7.6.5.35 `void cry_lld_start (CRYDriver * cryp)`

Configures and activates the crypto peripheral.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
----	-------------	---

Function Class:

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

7.6.5.36 void cry_lld_stop (CRYDriver * *cryp*)

Deactivates the crypto peripheral.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
----	-------------	---

Function Class:

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

7.6.5.37 cryerror_t cry_lld_loadkey (CRYDriver * *cryp*, cryalgorithm_t *algorithm*, size_t *size*, const uint8_t * *keyp*)

Initializes the transient key for a specific algorithm.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>algorithm</i>	the algorithm identifier
in	<i>size</i>	key size in bytes
in	<i>keyp</i>	pointer to the key data

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the specified algorithm is unknown or unsupported.
<i>CRY_ERR_INV_KEY_SIZE</i>	if the specified key size is invalid.

Function Class:

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

7.6.5.38 cryerror_t cry_lld_encrypt_AES (CRYDriver * *cryp*, crykey_t *key_id*, const uint8_t * *in*, uint8_t * *out*)

Encryption of a single block using AES.

Note

The implementation of this function must guarantee that it can be called from any context.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.39 `cryerror_t cry_lld_decrypt_AES (CRYDriver * cryp, crykey_t key_id, const uint8_t * in, uint8_t * out)`

Decryption of a single block using AES.

Note

The implementation of this function must guarantee that it can be called from any context.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.40 `cryerror_t cry_llc_encrypt_AES_ECB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out)`

Encryption operation using AES-ECB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of the selected key size
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.
<code>CRY_ERR_INV_KEY_TYPE</code>	the selected key is invalid for this operation.
<code>CRY_ERR_INV_KEY_ID</code>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.41 `cryerror_t cry_llc_decrypt_AES_ECB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out)`

Decryption operation using AES-ECB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
----	-------------	---

Parameters

in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of the selected key size
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.42 `cryerror_t cry_llc_encrypt_AES_CBC (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Encryption operation using AES-CBC.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of the selected key size
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits initial vector

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
--------------------	-----------------------------

Return values

<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.43 `cryerror_t cry_llc_decrypt_AES_CBC (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Decryption operation using AES-CBC.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of the selected key size
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits initial vector

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.44 `cryerror_t cry_llc_encrypt_AES_CFB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Encryption operation using AES-CFB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of the selected key size
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits initial vector

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.45 `cryerror_t cry_lld_decrypt_AES_CFB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Decryption operation using AES-CFB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of the selected key size
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits initial vector

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.46 `cryerror_t cry_ll_encrypt_AES_CTR (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Encryption operation using AES-CTR.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits initial vector + counter, it contains a 96 bits IV and a 32 bits counter

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.47 `cryerror_t cry_ll_decrypt_AES_CTR (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Decryption operation using AES-CTR.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext
in	<i>iv</i>	128 bits initial vector + counter, it contains a 96 bits IV and a 32 bits counter

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.48 `cryerror_t cry_ll_encrypt_AES_GCM (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv, size_t aadsize, const uint8_t * aad, uint8_t * authtag)`

Encryption operation using AES-GCM.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the text buffers, this number must be a multiple of 16
in	<i>in</i>	buffer containing the input plaintext

Parameters

out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	128 bits initial vector + counter, it contains a 96 bits IV and a 32 bits counter
in	<i>aadsize</i>	size of the authentication data, this number must be a multiple of 16
in	<i>aad</i>	buffer containing the authentication data
in	<i>authtag</i>	128 bits buffer for the generated authentication tag

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.49 `cryerror_t cry_ll_decrypt_AES_GCM (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv, size_t aadsize, const uint8_t * aad, uint8_t * authtag)`

Decryption operation using AES-GCM.

Note

The function operates on data buffers whose lenght is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the text buffers, this number must be a multiple of 16
in	<i>in</i>	buffer for the output cyphertext
out	<i>out</i>	buffer containing the input plaintext
in	<i>iv</i>	128 bits initial vector + counter, it contains a 96 bits IV and a 32 bits counter
in	<i>aadsize</i>	size of the authentication data, this number must be a multiple of 16
in	<i>aad</i>	buffer containing the authentication data
in	<i>authtag</i>	128 bits buffer for the generated authentication tag

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.50 `cryerror_t cry_llc_encrypt_DES (CRYDriver * cryp, crykey_t key_id, const uint8_t * in, uint8_t * out)`

Encryption of a single block using (T)DES.

Note

The implementation of this function must guarantee that it can be called from any context.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.51 `cryerror_t cry_llc_decrypt_DES (CRYDriver * cryp, crykey_t key_id, const uint8_t * in, uint8_t * out)`

Decryption of a single block using (T)DES.

Note

The implementation of this function must guarantee that it can be called from any context.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.52 `cryerror_t cry_lld_encrypt_DES_ECB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out)`

Encryption operation using (T)DES-ECB.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of 8
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

Return values

<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.53 `cryerror_t cry_lld_decrypt_DES_ECB (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out)`

Decryption operation using (T)DES-ECB.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔_id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of 8
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.54 `cryerror_t cry_lld_encrypt_DES_CBC (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Encryption operation using (T)DES-CBC.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of 8
in	<i>in</i>	buffer containing the input plaintext
out	<i>out</i>	buffer for the output cyphertext
in	<i>iv</i>	64 bits input vector

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.55 `cryerror_t cry_lld_decrypt_DES_CBC (CRYDriver * cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out, const uint8_t * iv)`

Decryption operation using (T)DES-CBC.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>key↔ _id</i>	the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
in	<i>size</i>	size of the plaintext buffer, this number must be a multiple of 8
in	<i>in</i>	buffer containing the input cyphertext
out	<i>out</i>	buffer for the output plaintext
in	<i>iv</i>	64 bits input vector

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.
<i>CRY_ERR_INV_KEY_TYPE</i>	the selected key is invalid for this operation.
<i>CRY_ERR_INV_KEY_ID</i>	if the specified key identifier is invalid or refers to an empty key slot.

Function Class:

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

7.6.5.56 `cryerror_t cry_llc_SHA1_init (CRYDriver * cryp, SHA1Context * sha1ctxp)`

Hash initialization using SHA1.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
out	<i>sha1ctxp</i>	pointer to a SHA1 context to be initialized

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.5.57 `cryerror_t cry_llc_SHA1_update (CRYDriver * cryp, SHA1Context * sha1ctxp, size_t size, const uint8_t * in)`

Hash update using SHA1.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha1ctxp</i>	pointer to a SHA1 context
in	<i>size</i>	size of input buffer
in	<i>in</i>	buffer containing the input text

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.5.58 `cryerror_t cry_lld_SHA1_final (CRYDriver * cryp, SHA1Context * sha1ctxp, uint8_t * out)`

Hash finalization using SHA1.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha1ctxp</i>	pointer to a SHA1 context
out	<i>out</i>	160 bits output buffer

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.5.59 `cryerror_t cry_lld_SHA256_init (CRYDriver * cryp, SHA256Context * sha256ctxp)`

Hash initialization using SHA256.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
out	<i>sha256ctxp</i>	pointer to a SHA256 context to be initialized

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.5.60 `cryerror_t cry_lld_SHA256_update (CRYDriver * cryp, SHA256Context * sha256ctxp, size_t size, const uint8_t * in)`

Hash update using SHA256.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha256ctxp</i>	pointer to a SHA256 context
in	<i>size</i>	size of input buffer
in	<i>in</i>	buffer containing the input text

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.5.61 `cryerror_t cry_lld_SHA256_final (CRYDriver * cryp, SHA256Context * sha256ctxp, uint8_t * out)`

Hash finalization using SHA256.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha256ctxp</i>	pointer to a SHA256 context
out	<i>out</i>	256 bits output buffer

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.5.62 **cryerror_t cry_lld_SHA512_init (CRYDriver * *cryp*, SHA512Context * *sha512ctxp*)**

Hash initialization using SHA512.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
out	<i>sha512ctxp</i>	pointer to a SHA512 context to be initialized

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.5.63 **cryerror_t cry_lld_SHA512_update (CRYDriver * *cryp*, SHA512Context * *sha512ctxp*, size_t *size*, const uint8_t * *in*)**

Hash update using SHA512.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha512ctxp</i>	pointer to a SHA512 context
in	<i>size</i>	size of input buffer
in	<i>in</i>	buffer containing the input text

Returns

The operation status.

Return values

<i>CRY_NOERROR</i>	if the operation succeeded.
<i>CRY_ERR_INV_ALGO</i>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.5.64 `cryerror_t cry_ild_SHA512_final (CRYDriver * cryp, SHA512Context * sha512ctxp, uint8_t * out)`

Hash finalization using SHA512.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
in	<i>sha512ctxp</i>	pointer to a SHA512 context
out	<i>out</i>	512 bits output buffer

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.5.65 `cryerror_t cry_ild_TRNG (CRYDriver * cryp, uint8_t * out)`

True random numbers generator.

Parameters

in	<i>cryp</i>	pointer to the CRYDriver object
out	<i>out</i>	128 bits output buffer

Returns

The operation status.

Return values

<code>CRY_NOERROR</code>	if the operation succeeded.
<code>CRY_ERR_INV_ALGO</code>	if the operation is unsupported on this device instance.

Function Class:

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

7.6.6 Variable Documentation

7.6.6.1 CRYDriver CRYD1

CRY1 driver identifier.

7.7 DAC Driver

Generic DAC Driver.

7.7.1 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](#) *dACP, [dacsample_t](#) *buffer, size_t n)
DAC notification callback type.
- typedef void(* [dacerrorcallback_t](#)) ([DACDriver](#) *dACP, [dacerror_t](#) err)
DAC 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](#) *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, [dacsample_t](#) *samples, size_t depth)
Starts a DAC conversion.
- void [dacStartConversionI](#) ([DACDriver](#) *dACP, const [DACConversionGroup](#) *grpp, [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, [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.

- void `dac_llt_init` (void)
Low level DAC driver initialization.
- void `dac_llt_start` (DACDriver *dacp)
Configures and activates the DAC peripheral.
- void `dac_llt_stop` (DACDriver *dacp)
Deactivates the DAC peripheral.
- void `dac_llt_put_channel` (DACDriver *dacp, dacchannel_t channel, dacsample_t sample)
Outputs a value directly on a DAC channel.
- void `dac_llt_start_conversion` (DACDriver *dacp)
Starts a DAC conversion.
- void `dac_llt_stop_conversion` (DACDriver *dacp)
Stops an ongoing conversion.

Enumerations

Variables

- DACDriver DACD1
DAC1 driver identifier.

7.7.2 Macro Definition Documentation

7.7.2.1 #define DAC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.7.2.2 #define DAC_USE_MUTUAL_EXCLUSION TRUE

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

Note

Disabling this option saves both code and data space.

7.7.2.3 #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 DACDriver object
----	-------------	---

Function Class:

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

7.7.2.4 `#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 DACDriver object
----	-------------	---

Function Class:

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

7.7.2.5 `#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 DACDriver object
----	-------------	---

Function Class:

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

7.7.2.6 `#define _dac_wakeup_isr(dacp)`

Value:

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

Wakes up the waiting thread.

Parameters

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

Function Class:

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

7.7.2.7 #define _dac_timeout_isr(*dacp*)**Value:**

```

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

```

Wakes up the waiting thread with a timeout message.

Parameters

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

Function Class:

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

7.7.2.8 #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	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.7.2.9 #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 DACDriver object
----	-------------	---

Function Class:

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

7.7.2.10 #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 DACDriver object
in	<i>err</i>	platform dependent error code

Function Class:

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

7.7.2.11 `#define DAC_MAX_CHANNELS 2`

Maximum number of DAC channels per unit.

7.7.2.12 `#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`.

7.7.3 Typedef Documentation

7.7.3.1 `typedef uint32_t dacchannel_t`

Type of a DAC channel index.

7.7.3.2 `typedef struct DACDriver DACDriver`

Type of a structure representing an DAC driver.

7.7.3.3 `typedef uint16_t dacsample_t`

Type representing a DAC sample.

7.7.3.4 `typedef void(* daccallback_t)(DACDriver *dacp, dacsample_t *buffer, size_t n)`

DAC notification callback type.

Parameters

in	<i>dacp</i>	pointer to the DACDriver 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 <code>buffer</code> callback

7.7.3.5 `typedef void(* dacerrorcallback_t)(DACDriver *dacp, dacerror_t err)`

DAC error callback type.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object triggering the callback
in	<i>err</i>	DAC error code

7.7.4 Enumeration Type Documentation

7.7.4.1 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.

7.7.4.2 enum dacerror_t

Possible DAC failure causes.

Note

Error codes are architecture dependent and should not be relied upon.

Enumerator

- DAC_ERR_DMAFAILURE*** DMA operations failure.
- DAC_ERR_UNDERFLOW*** DAC overflow condition.

7.7.5 Function Documentation

7.7.5.1 void dacInit (void)

DAC Driver initialization.

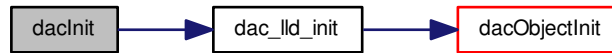
Note

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:



7.7.5.2 void dacObjectInit (DACDriver * *dacp*)

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

Parameters

out	<i>dacp</i>	pointer to the DACDriver 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:



7.7.5.3 void dacStart (DACDriver * *dacp*, const DACConfig * *config*)

Configures and activates the DAC peripheral.

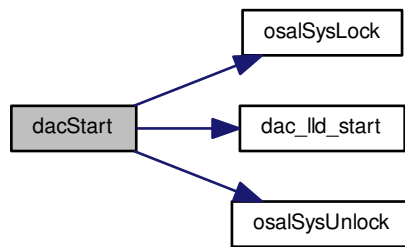
Parameters

in	<i>dacp</i>	pointer to the DACDriver object
in	<i>config</i>	pointer to the DACConfig object, it can be <code>NULL</code> 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:



7.7.5.4 void dacStop (DACDriver * *dacp*)

Deactivates the DAC peripheral.

Note

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

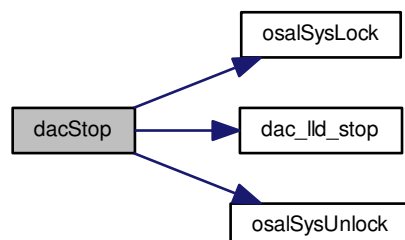
Parameters

in	<i>dacp</i>	pointer to the DACDriver 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:



7.7.5.5 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 DACDriver 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:



7.7.5.6 void dacStartConversion (DACDriver * *dacp*, const DACConversionGroup * *grpp*, 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.

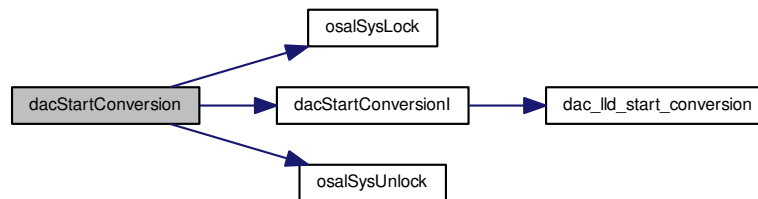
Parameters

in	<i>dacp</i>	pointer to the DACDriver object
in	<i>grpp</i>	pointer to a DACConversionGroup 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:



7.7.5.7 void `dacStartConversionI` (`DACDriver` * *dacp*, const `DACConversionGroup` * *grpp*, `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 <code>DACDriver</code> object
in	<i>grpp</i>	pointer to a <code>DACConversionGroup</code> 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:



7.7.5.8 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.

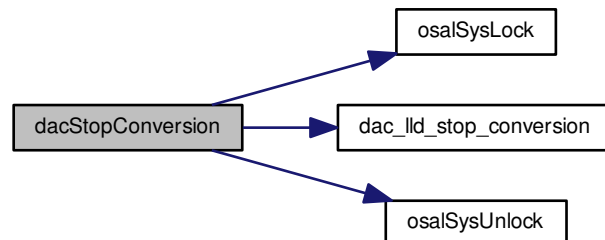
Parameters

in	<i>dacp</i>	pointer to the <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:



7.7.5.9 void dacStopConversionI (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.

Here is the call graph for this function:



7.7.5.10 `msg_t dacConvert (DACDriver * dacp, const DACConversionGroup * grpp, 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 DACDriver object
in	<i>grpp</i>	pointer to a DACConversionGroup 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.

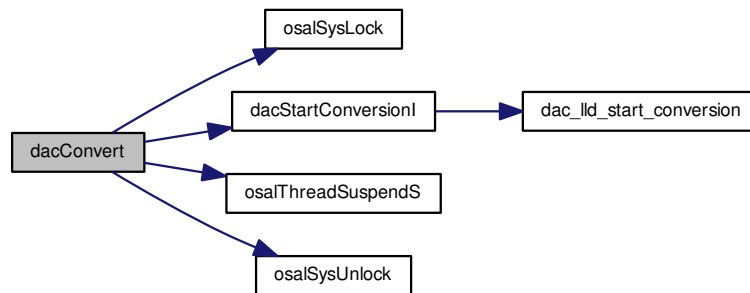
Return values

<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:



7.7.5.11 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 DACDriver 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:



7.7.5.12 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 DACDriver 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:

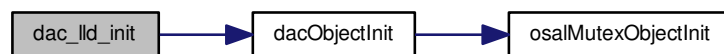
**7.7.5.13 void dac_lld_init (void)**

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:

**7.7.5.14 void dac_lld_start (DACDriver * dacp)**

Configures and activates the DAC peripheral.

Parameters

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

Function Class:

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

7.7.5.15 void dac_ild_stop (DACDriver * *dacp*)

Deactivates the DAC peripheral.

Parameters

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

Function Class:

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

7.7.5.16 void dac_ild_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 DACDriver 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.

7.7.5.17 void dac_ild_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 DACDriver object
----	-------------	---------------------------------

Function Class:

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

7.7.5.18 void dac_lld_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 DACDriver 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.

7.7.6 Variable Documentation

7.7.6.1 DACDriver DACD1

DAC1 driver identifier.

7.8 EXT Driver

Generic EXT Driver.

7.8.1 Detailed Description

Generic EXT Driver.

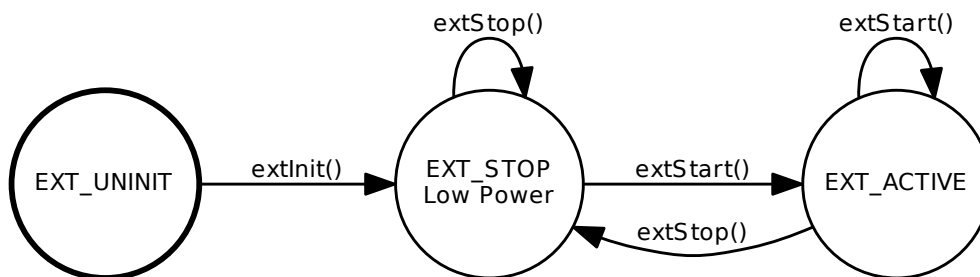
This module implements a generic EXT (EXternal) driver.

Precondition

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

7.8.2 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).



7.8.3 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_LOW_LEVEL` 5U
low level callback.
- #define `EXT_CH_MODE_AUTOSTART` 4U
Channel started automatically on driver start.

Macro Functions

- #define `extChannelEnable`(extp, channel) `ext_ild_channel_enable`(extp, channel)
Enables an EXT channel.
- #define `extChannelDisable`(extp, channel) `ext_ild_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_llc_init` (void)
Low level EXT driver initialization.
- void `ext_llc_start` (EXTDriver *extp)
Configures and activates the EXT peripheral.
- void `ext_llc_stop` (EXTDriver *extp)
Deactivates the EXT peripheral.
- void `ext_llc_channel_enable` (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void `ext_llc_channel_disable` (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.

Enumerations

Variables

- `EXTDriver EXT1`
EXT1 driver identifier.

7.8.4 Macro Definition Documentation

7.8.4.1 #define EXT_CH_MODE_EDGES_MASK 3U

Mask of edges field.

7.8.4.2 #define EXT_CH_MODE_DISABLED 0U

Channel disabled.

7.8.4.3 #define EXT_CH_MODE_RISING_EDGE 1U

Rising edge callback.

7.8.4.4 #define EXT_CH_MODE_FALLING_EDGE 2U

Falling edge callback.

7.8.4.5 #define EXT_CH_MODE_BOTH_EDGES 3U

Both edges callback.

7.8.4.6 #define EXT_CH_MODE_LOW_LEVEL 5U

low level callback.

7.8.4.7 #define EXT_CH_MODE_AUTOSTART 4U

Channel started automatically on driver start.

7.8.4.8 #define extChannelEnable(*extp, channel*) ext_lld_channel_enable(extp, channel)

Enables an EXT channel.

Parameters

in	<i>extp</i>	pointer to the EXTDriver 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.

7.8.4.9 #define extChannelDisable(*extp, channel*) ext_lld_channel_disable(extp, channel)

Disables an EXT channel.

Parameters

in	<i>extp</i>	pointer to the EXTDriver 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.

7.8.4.10 #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.

7.8.4.11 #define EXT_MAX_CHANNELS 20

Available number of EXT channels.

7.8.4.12 #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`.

7.8.5 Typedef Documentation**7.8.5.1 typedef struct EXTDriver EXTDriver**

Type of a structure representing a EXT driver.

7.8.5.2 typedef uint32_t expchannel_t

EXT channel identifier.

7.8.5.3 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>EXPDriver</code> object triggering the callback
----	-------------	--

7.8.6 Enumeration Type Documentation

7.8.6.1 enum extstate_t

Driver state machine possible states.

Enumerator

EXT_UNINIT Not initialized.

EXT_STOP Stopped.

EXT_ACTIVE Active.

7.8.7 Function Documentation

7.8.7.1 void extInit (void)

EXT Driver initialization.

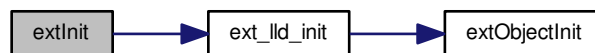
Note

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:



7.8.7.2 void extObjectInit (EXTDriver * extp)

Initializes the standard part of a `EXTDriver` structure.

Parameters

out	<i>extp</i>	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.

7.8.7.3 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.

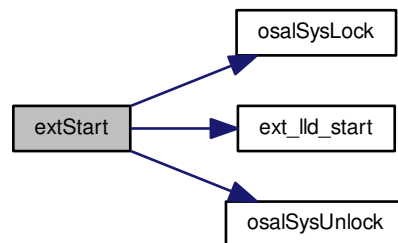
Parameters

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

**7.8.7.4 void extStop (EXTDriver * extp)**

Deactivates the EXT peripheral.

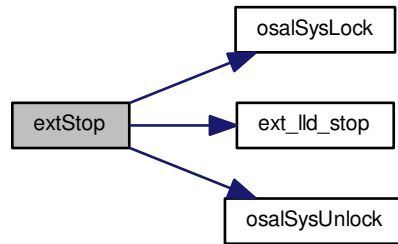
Parameters

in	<i>extp</i>	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:



7.8.7.5 void extChannelEnable (EXTDriver * extp, expchannel_t channel)

Enables an EXT channel.

Precondition

The channel must not be in `EXT_CH_MODE_DISABLED` mode.

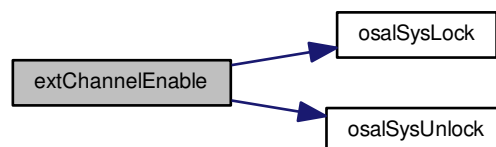
Parameters

in	<i>extp</i>	pointer to the EXTDriver 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:



7.8.7.6 void extChannelDisable (EXTDriver * extp, expchannel_t channel)

Disables an EXT channel.

Precondition

The channel must not be in `EXT_CH_MODE_DISABLED` mode.

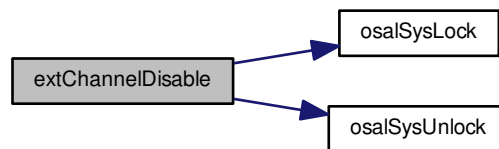
Parameters

in	<i>extp</i>	pointer to the EXTDriver 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:



7.8.7.7 `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	<i>extp</i>	pointer to the EXTDriver object
in	<i>channel</i>	channel to be changed
in	<i>extcp</i>	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:



7.8.7.8 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:



7.8.7.9 void ext_lld_start (EXTDriver * extp)

Configures and activates the EXT peripheral.

Parameters

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

Function Class:

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

7.8.7.10 void ext_lld_stop (EXTDriver * extp)

Deactivates the EXT peripheral.

Parameters

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

Function Class:

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

7.8.7.11 void ext_lld_channel_enable (EXTDriver * *extp*, expchannel_t *channel*)

Enables an EXT channel.

Parameters

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

Function Class:

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

7.8.7.12 void ext_lld_channel_disable (EXTDriver * *extp*, expchannel_t *channel*)

Disables an EXT channel.

Parameters

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

Function Class:

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

7.8.8 Variable Documentation

7.8.8.1 EXTDriver EXT1

EXT1 driver identifier.

7.9 Generic Flash Driver Interface

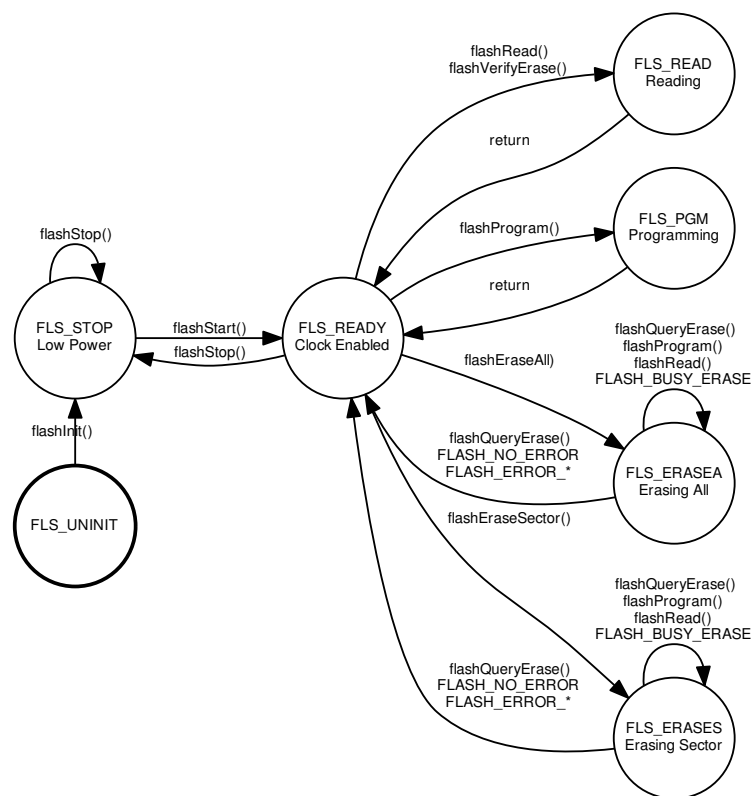
HAL Generic Flash Driver Interface.

7.9.1 Detailed Description

HAL Generic Flash Driver Interface.

7.9.2 Driver State Machine

The flash 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 _base_flash_methods_alone`
BaseFlash specific methods.
- `#define _base_flash_methods`
BaseFlash specific methods with inherited ones.
- `#define _base_flash_data`
BaseFlash specific data.

Flash attributes

- #define **FLASH_ATTR_ERASED_IS_ONE** 0x00000001
- #define **FLASH_ATTR_MEMORY_MAPPED** 0x00000002
- #define **FLASH_ATTR_REWRITABLE** 0x00000004
- #define **FLASH_ATTR_READ_ECC_CAPABLE** 0x00000008
- #define **FLASH_ATTR_SUSPEND_ERASE_CAPABLE** 0x00000010

Macro Functions (BaseFlash)

- #define **getBaseFlash**(ip) ((BaseFlash *)&(ip)->vmt)
Instance getter.
- #define **flashGetDescriptor**(ip) (ip)->vmt->get_descriptor(ip)
Sensors get axes number.
- #define **flashRead**(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)
Read operation.
- #define **flashProgram**(ip, offset, n, pp) (ip)->vmt->program(ip, offset, n, pp)
Program operation.
- #define **flashStartEraseAll**(ip) (ip)->vmt->start_erase_all(ip)
Starts a whole-device erase operation.
- #define **flashStartEraseSector**(ip, sector) (ip)->vmt->start_erase_sector(ip, sector)
Starts an sector erase operation.
- #define **flashQueryErase**(ip, msec) (ip)->vmt->query_erase(ip, msec)
Queries the driver for erase operation progress.
- #define **flashVerifyErase**(ip, sector) (ip)->vmt->verify_erase(ip, sector)
Returns the erase state of a sector.

Typedefs

- typedef uint32_t **flash_offset_t**
Type of a flash offset.
- typedef uint32_t **flash_sector_t**
Type of a flash sector number.

Data Structures

- struct **flash_sector_descriptor_t**
Flash sector descriptor.
- struct **flash_descriptor_t**
Type of a flash device descriptor.
- struct **BaseFlashVMT**
BaseFlash virtual methods table.
- struct **BaseFlash**
Base flash class.

Functions

- **flash_error_t** **flashWaitErase** (BaseFlash *devp)
Waits until the current erase operation is finished.
- **flash_offset_t** **flashGetSectorOffset** (BaseFlash *devp, flash_sector_t sector)
Returns the offset of a sector.
- **uint32_t** **flashGetSectorSize** (BaseFlash *devp, flash_sector_t sector)
Returns the size of a sector.

Enumerations

7.9.3 Macro Definition Documentation

7.9.3.1 #define _base_flash_methods_alone

Value:

```
/* Get flash device attributes.*/
const flash_descriptor_t * (*get_descriptor)(void *instance);
/* Read operation.*/
flash_error_t (*read)(void *instance, flash_offset_t offset,
                      size_t n, uint8_t *rp);
/* Program operation.*/
flash_error_t (*program)(void *instance, flash_offset_t offset,
                        size_t n, const uint8_t *pp);
/* Erase whole flash device.*/
flash_error_t (*start_erase_all)(void *instance);
/* Erase single sector.*/
flash_error_t (*start_erase_sector)(void *instance,
                                   \
                                   flash_sector_t
                                   sector);
flash_error_t (*query_erase)(void *instance, uint32_t *wait_time);
/* Verify erase single sector.*/
flash_error_t (*verify_erase)(void *instance, flash_sector_t sector);
```

BaseFlash specific methods.

Note

No methods so far, just a common ancestor interface.

7.9.3.2 #define _base_flash_methods

Value:

```
_base_object_methods
_base_flash_methods_alone
```

BaseFlash specific methods with inherited ones.

7.9.3.3 #define _base_flash_data

Value:

```
_base_object_data
/* Driver state.*/
flash_state_t state;
```

BaseFlash specific data.

7.9.3.4 #define getBaseFlash(ip)((BaseFlash *)&(ip)->vmt)

Instance getter.

This special method is used to get the instance of this class object from a derived class.

7.9.3.5 #define flashGetDescriptor(ip)(ip)->vmt->get_descriptor(ip)

Sensors get axes number.

Parameters

in	<i>ip</i>	pointer to a BaseFlash or derived class
----	-----------	---

Returns

A flash device descriptor.

Function Class:

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

7.9.3.6 `#define flashRead(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)`

Read operation.

Parameters

in	<i>ip</i>	pointer to a BaseFlash or derived class
in	<i>offset</i>	flash offset
in	<i>n</i>	number of bytes to be read
out	<i>rp</i>	pointer to the data buffer

Returns

An error code.

Return values

<i>FLASH_NO_ERROR</i>	if there is no erase operation in progress.
<i>FLASH_BUSY_ERASING</i>	if there is an erase operation in progress.
<i>FLASH_ERROR_READ</i>	if the read operation failed.
<i>FLASH_ERROR_HW_FAILURE</i>	if access to the memory failed.

Function Class:

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

7.9.3.7 `#define flashProgram(ip, offset, n, pp) (ip)->vmt->program(ip, offset, n, pp)`

Program operation.

Parameters

in	<i>ip</i>	pointer to a BaseFlash or derived class
in	<i>offset</i>	flash offset
in	<i>n</i>	number of bytes to be programmed
in	<i>pp</i>	pointer to the data buffer

Returns

An error code.

Return values

<i>FLASH_NO_ERROR</i>	if there is no erase operation in progress.
<i>FLASH_BUSY_ERASING</i>	if there is an erase operation in progress.
<i>FLASH_ERROR_PROGRAM</i>	if the program operation failed.
<i>FLASH_ERROR_HW_FAILURE</i>	if access to the memory failed.

Function Class:

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

7.9.3.8 #define flashStartEraseAll(ip) (ip)->vmt->start_erase_all(ip)

Starts a whole-device erase operation.

Parameters

in	<i>ip</i>	pointer to a BaseFlash or derived class
----	-----------	---

Returns

An error code.

Return values

<i>FLASH_NO_ERROR</i>	if there is no erase operation in progress.
<i>FLASH_BUSY_ERASING</i>	if there is an erase operation in progress.
<i>FLASH_ERROR_HW_FAILURE</i>	if access to the memory failed.

Function Class:

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

7.9.3.9 #define flashStartEraseSector(ip, sector) (ip)->vmt->start_erase_sector(ip, sector)

Starts an sector erase operation.

Parameters

in	<i>ip</i>	pointer to a BaseFlash or derived class
in	<i>sector</i>	sector to be erased

Returns

An error code.

Return values

<i>FLASH_NO_ERROR</i>	if there is no erase operation in progress.
<i>FLASH_BUSY_ERASING</i>	if there is an erase operation in progress.
<i>FLASH_ERROR_HW_FAILURE</i>	if access to the memory failed.

Function Class:

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

7.9.3.10 #define flashQueryErase(ip, msec)(ip)->vmt->query_erase(ip, msec)

Queries the driver for erase operation progress.

Parameters

in	<i>ip</i>	pointer to a BaseFlash or derived class
out	<i>msec</i>	recommended time, in milliseconds, that what should be spent before calling this function again, can be <code>NULL</code>

Returns

An error code.

Return values

<i>FLASH_NO_ERROR</i>	if there is no erase operation in progress.
<i>FLASH_BUSY_ERASING</i>	if there is an erase operation in progress.
<i>FLASH_ERROR_ERASE</i>	if the erase operation failed.
<i>FLASH_ERROR_HW_FAILURE</i>	if access to the memory failed.

Function Class:

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

7.9.3.11 #define flashVerifyErase(ip, sector)(ip)->vmt->verify_erase(ip, sector)

Returns the erase state of a sector.

Parameters

in	<i>ip</i>	pointer to a BaseFlash or derived class
in	<i>sector</i>	sector to be verified

Returns

An error code.

Return values

<i>FLASH_NO_ERROR</i>	if the sector is erased.
<i>FLASH_BUSY_ERASING</i>	if there is an erase operation in progress.
<i>FLASH_ERROR_VERIFY</i>	if the verify operation failed.
<i>FLASH_ERROR_HW_FAILURE</i>	if access to the memory failed.

Function Class:

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

7.9.4 Typedef Documentation

7.9.4.1 typedef uint32_t flash_offset_t

Type of a flash offset.

7.9.4.2 typedef uint32_t flash_sector_t

Type of a flash sector number.

7.9.5 Enumeration Type Documentation

7.9.5.1 enum flash_state_t

Driver state machine possible states.

7.9.5.2 enum flash_error_t

Type of a flash error code.

7.9.6 Function Documentation

7.9.6.1 flash_error_t flashWaitErase (BaseFlash * devp)

Waits until the current erase operation is finished.

Parameters

in	<i>devp</i>	pointer to a BaseFlash object
----	-------------	---

Returns

An error code.

Return values

<i>FLASH_NO_ERROR</i>	if there is no erase operation in progress.
<i>FLASH_ERROR_ERASE</i>	if the erase operation failed.
<i>FLASH_ERROR_HW_FAILURE</i>	if access to the memory failed.

7.9.6.2 `flash_offset_t flashGetSectorOffset (BaseFlash * devp, flash_sector_t sector)`

Returns the offset of a sector.

Parameters

in	<i>devp</i>	pointer to a BaseFlash object
in	<i>sector</i>	flash sector number

Returns

the offset of the sector

7.9.6.3 `uint32_t flashGetSectorSize (BaseFlash * devp, flash_sector_t sector)`

Returns the size of a sector.

Parameters

in	<i>devp</i>	pointer to a BaseFlash object
in	<i>sector</i>	flash sector number

Returns

the size of the sector

7.10 GPT Driver

Generic GPT Driver.

7.10.1 Detailed Description

Generic GPT Driver.

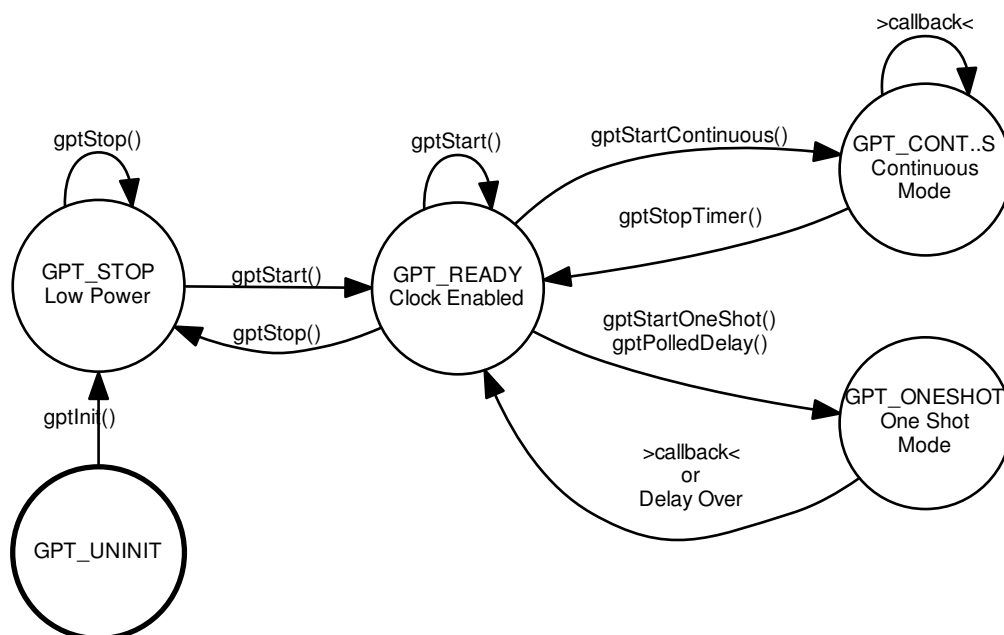
This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

Precondition

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

7.10.2 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).



7.10.3 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_llc_get_interval(gptp)`
Returns the interval of GPT peripheral.
- #define `gptGetCounterX(gptp) gpt_llc_get_counter(gptp)`
Returns the counter value of GPT peripheral.
- #define `gpt_llc_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

Variables

- `GPTDriver GPTD1`
GPTD1 driver identifier.

7.10.4 Macro Definition Documentation

7.10.4.1 #define gptChangeIntervall(*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 GPTDriver 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.

7.10.4.2 #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 GPTDriver object
----	-------------	---

Returns

The current interval.

Function Class:

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

7.10.4.3 #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>gtp</i>	pointer to a GPTDriver object
----	------------	---

Returns

The current counter value.

Function Class:

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

7.10.4.4 #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`.

7.10.4.5 #define gpt_lld_change_interval(gtp, interval)**Value:**

```
{
    (void)gtp;
    (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>gtp</i>	pointer to a GPTDriver object
in	<i>interval</i>	new cycle time in timer ticks

Function Class:

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

7.10.5 Typedef Documentation

7.10.5.1 typedef struct GPTDriver GPTDriver

Type of a structure representing a GPT driver.

7.10.5.2 typedef void(* gptcallback_t) (GPTDriver *gtp)

GPT notification callback type.

Parameters

in	<i>gtp</i>	pointer to a GPTDriver object
----	------------	---

7.10.5.3 typedef uint32_t gptfreq_t

GPT frequency type.

7.10.5.4 typedef uint16_t gptcnt_t

GPT counter type.

7.10.6 Enumeration Type Documentation

7.10.6.1 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.

7.10.7 Function Documentation

7.10.7.1 void gptlnit (void)

GPT Driver initialization.

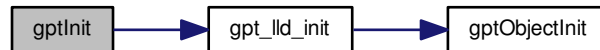
Note

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:

**7.10.7.2 void gptObjectInit (GPTDriver * gtp)**

Initializes the standard part of a `GPTDriver` structure.

Parameters

out	<i>gtp</i>	pointer to the <code>GPTDriver</code> object
-----	------------	--

Function Class:

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

7.10.7.3 void gptStart (GPTDriver * gtp, const GPTConfig * config)

Configures and activates the GPT peripheral.

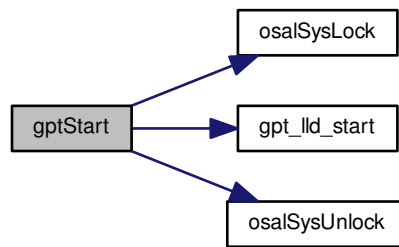
Parameters

in	<i>gtp</i>	pointer to the <code>GPTDriver</code> object
in	<i>config</i>	pointer to the <code>GPTConfig</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:



7.10.7.4 void gptStop (GPTDriver * gptp)

Deactivates the GPT peripheral.

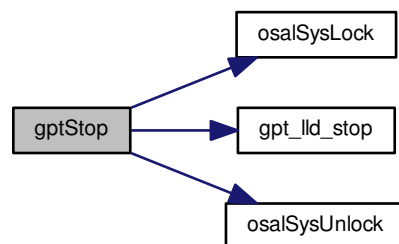
Parameters

in	<i>gptp</i>	pointer to the <code>GPTDriver</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:



7.10.7.5 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.

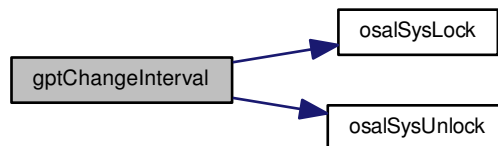
Parameters

in	<i>gtp</i>	pointer to a GPTDriver 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:

**7.10.7.6 void gptStartContinuous (GPTDriver * gtp, gptcnt_t interval)**

Starts the timer in continuous mode.

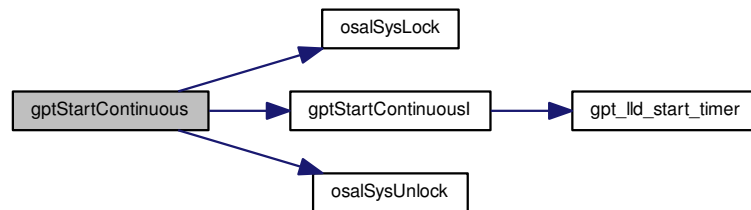
Parameters

in	<i>gtp</i>	pointer to the GPTDriver 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:



7.10.7.7 void gptStartContinuousI (GPTDriver * gptp, gptcnt_t interval)

Starts the timer in continuous mode.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver 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:



7.10.7.8 void gptStartOneShot (GPTDriver * gptp, gptcnt_t interval)

Starts the timer in one shot mode.

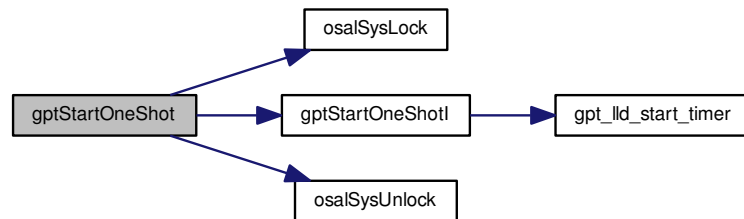
Parameters

in	<i>gptp</i>	pointer to the GPTDriver 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:



7.10.7.9 void gptStartOneShotI (GPTDriver * gptp, gptcnt_t interval)

Starts the timer in one shot mode.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver 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:



7.10.7.10 void gptStopTimer (GPTDriver * gptp)

Stops the timer.

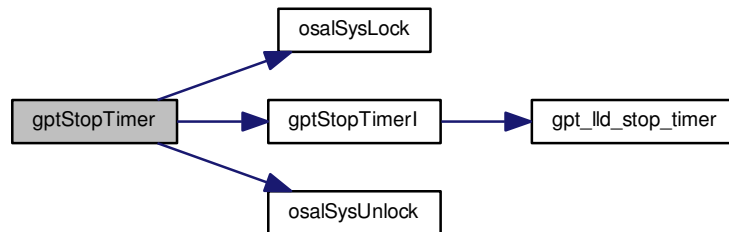
Parameters

in	<i>gptp</i>	pointer to the GPTDriver 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:



7.10.7.11 void gptStopTimerI (GPTDriver * gptp)

Stops the timer.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver 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:



7.10.7.12 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>gtp</i>	pointer to the GPTDriver 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:

**7.10.7.13 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:

**7.10.7.14 void gpt_lld_start (GPTDriver * gtp)**

Configures and activates the GPT peripheral.

Parameters

in	<i>gtp</i>	pointer to the GPTDriver object
----	------------	---

Function Class:

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

7.10.7.15 void gpt_ild_stop (GPTDriver * *gtp*)

Deactivates the GPT peripheral.

Parameters

in	<i>gtp</i>	pointer to the GPTDriver object
----	------------	---

Function Class:

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

7.10.7.16 void gpt_ild_start_timer (GPTDriver * *gtp*, gptcnt_t *interval*)

Starts the timer in continuous mode.

Parameters

in	<i>gtp</i>	pointer to the GPTDriver object
in	<i>interval</i>	period in ticks

Function Class:

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

7.10.7.17 void gpt_ild_stop_timer (GPTDriver * *gtp*)

Stops the timer.

Parameters

in	<i>gtp</i>	pointer to the GPTDriver object
----	------------	---

Function Class:

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

7.10.7.18 void gpt_ild_polled_delay (GPTDriver * *gtp*, 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>gtp</i>	pointer to the GPTDriver object
in	<i>interval</i>	time interval in ticks

Function Class:

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

7.10.8 Variable Documentation**7.10.8.1 GPTDriver GPTD1**

GPTD1 driver identifier.

7.11 Generic Gyroscope Interface

HAL Generic Gyroscope Interface.

7.11.1 Detailed Description

HAL Generic Gyroscope Interface.

Macros

- `#define _base_gyroscope_methods_alone`
BaseGyroscope specific methods.
- `#define _base_gyroscope_methods`
BaseGyroscope specific methods with inherited ones.
- `#define _base_gyroscope_data_base_sensor_data`
BaseGyroscope specific data.

Macro Functions (BaseGyroscope)

- `#define gyroscopeGetAxesNumber(ip) (ip)->vmt->get_channels_number(ip)`
Gyroscope get axes number.
- `#define gyroscopeReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)`
Gyroscope read raw data.
- `#define gyroscopeReadCooked(ip, dp) (ip)->vmt->read_cooked(ip, dp)`
Gyroscope read cooked data.
- `#define gyroscopeSampleBias(ip) (ip)->vmt->sample_bias(ip)`
Gyroscope bias sampling procedure.
- `#define gyroscopeSetBias(ip, bp) (ip)->vmt->set_bias(ip, bp)`
Updates gyroscope bias data from received buffer.
- `#define gyroscopeResetBias(ip) (ip)->vmt->reset_bias(ip)`
Reset gyroscope bias data restoring it to zero.
- `#define gyroscopeSetSensitivity(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)`
Updates gyroscope sensitivity data from received buffer.
- `#define gyroscopeResetSensitivity(ip) (ip)->vmt->reset_sensitivity(ip)`
Reset gyroscope sensitivity data restoring it to its typical value.

Data Structures

- struct `BaseGyroscopeVMT`
BaseGyroscope virtual methods table.
- struct `BaseGyroscope`
Base gyroscope class.

7.11.2 Macro Definition Documentation

7.11.2.1 `#define _base_gyroscope_methods_alone`

Value:

```

/* Invoke the sample bias procedure.*/
msg_t (*sample_bias)(void *instance);
/* Invoke the set bias procedure.*/
msg_t (*set_bias)(void *instance, float biases[]);
/* Remove bias stored data.*/
msg_t (*reset_bias)(void *instance);
/* Invoke the set sensitivity procedure.*/
msg_t (*set_sensitivity)(void *instance, float sensitivities[]);
/* Restore sensitivity stored data to default.*/
msg_t (*reset_sensitivity)(void *instance);

```

[BaseGyroscope](#) specific methods.

7.11.2.2 #define _base_gyroscope_methods

Value:

```

_base_sensor_methods
_base_gyroscope_methods_alone

```

[BaseGyroscope](#) specific methods with inherited ones.

7.11.2.3 #define _base_gyroscope_data_base_sensor_data

[BaseGyroscope](#) specific data.

7.11.2.4 #define gyroscopeGetAxesNumber(ip) (ip)->vmt->get_channels_number(ip)

Gyroscope get axes number.

Parameters

in	<i>ip</i>	pointer to a BaseGyroscope class.
----	-----------	---

Returns

The number of axes of the [BaseGyroscope](#)

Function Class:

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

7.11.2.5 #define gyroscopeReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)

Gyroscope read raw data.

Parameters

in	<i>ip</i>	pointer to a BaseGyroscope class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.11.2.6 `#define gyroscopeReadCooked(ip, dp)(ip)->vmt->read_cooked(ip, dp)`

Gyroscope read cooked data.

Parameters

in	<i>ip</i>	pointer to a BaseGyroscope class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.11.2.7 `#define gyroscopeSampleBias(ip)(ip)->vmt->sample_bias(ip)`

Gyroscope bias sampling procedure.

Note

During this procedure gyroscope must be kept hold in the rest position. Sampled bias will be automatically removed after calling this procedure.

Parameters

in	<i>ip</i>	pointer to a BaseGyroscope class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.11.2.8 `#define gyroscopeSetBias(ip, bp)(ip)->vmt->set_bias(ip, bp)`

Updates gyroscope bias data from received buffer.

Note

The bias buffer must have the same length of the the gyroscope axes number.

Parameters

in	<i>ip</i>	pointer to a BaseGyroscope class.
in	<i>bp</i>	pointer to a buffer of bias values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.11.2.9 `#define gyroscopeResetBias(ip)(ip)->vmt->reset_bias(ip)`

Reset gyroscope bias data restoring it to zero.

Parameters

in	<i>ip</i>	pointer to a BaseGyroscope class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.11.2.10 `#define gyroscopeSetSensitivity(ip, sp)(ip)->vmt->set_sensitivity(ip, sp)`

Updates gyroscope sensitivity data from received buffer.

Note

The sensitivity buffer must have the same length of the the gyroscope axes number.

Parameters

in	<i>ip</i>	pointer to a BaseGyroscope class.
in	<i>sp</i>	pointer to a buffer of sensitivity values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.11.2.11 `#define gyroscopeResetSensitivity(ip)(ip)->vmt->reset_sensitivity(ip)`

Reset gyroscope sensitivity data restoring it to its typical value.

Parameters

in	<i>ip</i>	pointer to a BaseGyroscope class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.12 HAL Driver

Hardware Abstraction Layer.

7.12.1 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 "6.0.1"`
HAL version string.
- `#define CH_HAL_MAJOR 6`
HAL version major number.
- `#define CH_HAL_MINOR 0`
HAL version minor number.
- `#define CH_HAL_PATCH 1`
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.

7.12.2 Macro Definition Documentation

7.12.2.1 `#define _CHIBIOS_HAL_`

ChibiOS/HAL identification macro.

7.12.2.2 `#define CH_HAL_STABLE 1`

Stable release flag.

7.12.2.3 `#define HAL_VERSION "6.0.1"`

HAL version string.

7.12.2.4 `#define CH_HAL_MAJOR 6`

HAL version major number.

7.12.2.5 `#define CH_HAL_MINOR 0`

HAL version minor number.

7.12.2.6 `#define CH_HAL_PATCH 1`

HAL version patch number.

7.12.3 Function Documentation

7.12.3.1 `void hallnit (void)`

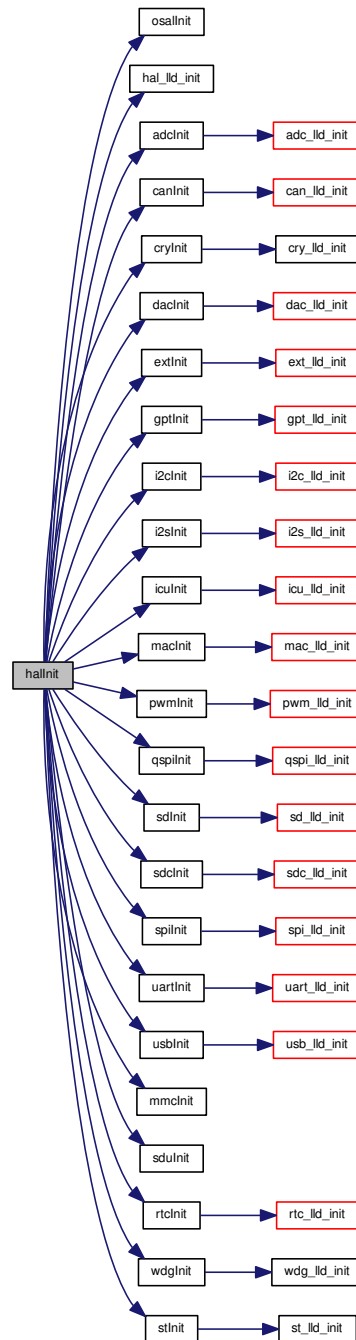
HAL initialization.

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:



7.12.3.2 void hal_ild_init (void)

Low level HAL driver initialization.

Function Class:

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

7.13 I/O Buffers Queues

7.13.1 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 bqIsSuspendedX(bqp) ((bqp)->suspended)`
Return the suspended state of the queue.
- `#define bqSuspendI(bqp)`
Puts the queue in suspended state.
- `#define bqResumeX(bqp)`
Resumes normal queue operations.
- `#define ibqlsEmptyI(ibqp) ((bool)(bqSpacel(ibqp) == 0U))`
Evaluates to `true` if the specified input buffers queue is empty.
- `#define ibqlsFullI(ibqp)`
Evaluates to `true` if the specified input buffers queue is full.
- `#define obqlsEmptyI(obqp)`
Evaluates to `true` if the specified output buffers queue is empty.
- `#define obqlsFullI(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, bool suspended, [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, [sysinterval_t](#) timeout)
Gets the next filled buffer from the queue.
- [msg_t](#) [ibqGetFullBufferTimeoutS](#) ([input_buffers_queue_t](#) *ibqp, [sysinterval_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, [sysinterval_t](#) timeout)
Input queue read with timeout.
- [size_t](#) [ibqReadTimeout](#) ([input_buffers_queue_t](#) *ibqp, [uint8_t](#) *bp, [size_t](#) n, [sysinterval_t](#) timeout)
Input queue read with timeout.
- void [obqObjectInit](#) ([output_buffers_queue_t](#) *obqp, bool suspended, [uint8_t](#) *bp, [size_t](#) size, [size_t](#) n, [bqnotify_t](#) only, 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, [sysinterval_t](#) timeout)
Gets the next empty buffer from the queue.
- [msg_t](#) [obqGetEmptyBufferTimeoutS](#) ([output_buffers_queue_t](#) *obqp, [sysinterval_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, [sysinterval_t](#) timeout)
Output queue write with timeout.
- [size_t](#) [obqWriteTimeout](#) ([output_buffers_queue_t](#) *obqp, const [uint8_t](#) *bp, [size_t](#) n, [sysinterval_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.

7.13.2 Macro Definition Documentation

7.13.2.1 `#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

7.13.2.2 `#define bqSizeX(bqp) ((bqp)->bn)`

Returns the queue's number of buffers.

Parameters

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

Returns

The number of buffers.

Function Class:

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

7.13.2.3 `#define bqSpacel(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 <code>io_buffers_queue_t</code> 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.

7.13.2.4 `#define bqGetLinkX(bqp) ((bqp)->link)`

Returns the queue application-defined link.

Parameters

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

Returns

The application-defined link.

Function Class:

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

7.13.2.5 #define bqIsSuspendedX(*bqp*)((bqp)->suspended)

Return the suspended state of the queue.

Parameters

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

Returns

The suspended state.

Return values

<i>false</i>	if blocking access to the queue is enabled.
<i>true</i>	if blocking access to the queue is suspended.

Function Class:

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

7.13.2.6 #define bqSuspendI(*bqp*)**Value:**

```
{
    (bqp)->suspended = true;
    osalThreadDequeueAllI (&(bqp)->waiting, MSG_RESET);
}
```

Puts the queue in suspended state.

When the queue is put in suspended state all waiting threads are woken with message `MSG_RESET` and subsequent attempt at waiting on the queue will result in an immediate return with `MSG_RESET` message.

Note

The content of the queue is not altered, queues can be accessed is suspended state until a blocking operation is met then a `MSG_RESET` occurs.

Parameters

in	<i>bqp</i>	pointer to an <code>io_buffers_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.

7.13.2.7 #define bqResumeX(*bqp*)**Value:**

```
{
    (bqp)->suspended = false;
}
```

Resumes normal queue operations.

Parameters

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

Function Class:

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

7.13.2.8 #define ibqlsEmpty(*ibqp*) ((bool)(bqSpacel(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.

7.13.2.9 #define ibqlsFull(*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.

7.13.2.10 #define obqlsEmpty(*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.

7.13.2.11 `#define obqIsFull(obqp) ((bool)(bqSpacel(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.

7.13.3 Typedef Documentation

7.13.3.1 `typedef struct io_buffers_queue io_buffers_queue_t`

Type of a generic queue of buffers.

7.13.3.2 `typedef void(* bqnotify_t) (io_buffers_queue_t *bqp)`

Double buffer notification callback type.

Parameters

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

7.13.3.3 `typedef io_buffers_queue_t input_buffers_queue_t`

Type of an input buffers queue.

7.13.3.4 `typedef io_buffers_queue_t output_buffers_queue_t`

Type of an output buffers queue.

7.13.4 Function Documentation

7.13.4.1 `void ibqObjectInit (input_buffers_queue_t * ibqp, bool suspended, 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>suspended</i>	initial state of the queue
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:



7.13.4.2 `void ibqResetl (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:



7.13.4.3 uint8_t* ibqGetEmptyBufferI (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 <code>input_buffers_queue_t</code> object
----	-------------	--

Returns

A pointer to the next buffer to be filled.

Return values

<code>NULL</code>	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.

7.13.4.4 void ibqPostFullBufferI (input_buffers_queue_t * *ibqp*, size_t *size*)

Posts a new filled buffer to the queue.

Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> 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:



7.13.4.5 msg_t ibqGetFullBufferTimeout (input_buffers_queue_t * *ibqp*, sysinterval_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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

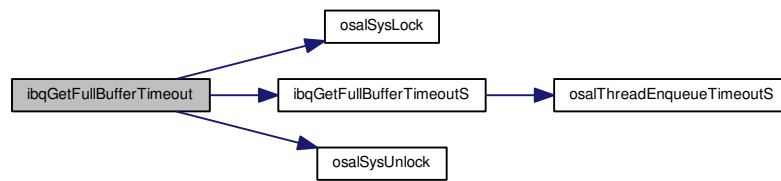
Return values

<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 or has been put in suspended 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:



7.13.4.6 msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t * *ibqp*, sysinterval_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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

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 or has been put in suspended state.

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:



7.13.4.7 void ibqReleaseEmptyBuffer (input_buffers_queue_t * ibqp)

Releases the buffer back in the queue.

Note

The object callback is called after releasing the buffer.

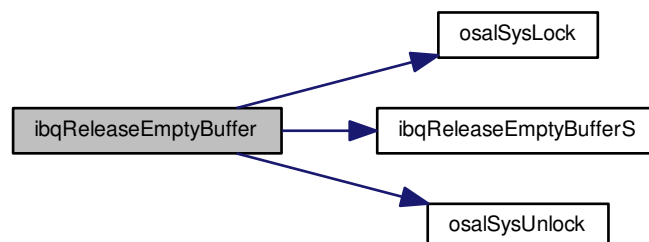
Parameters

in	<i>ibqp</i>	pointer to the <code>input_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:



7.13.4.8 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.

7.13.4.9 `msg_t ibqGetTimeout(input_buffers_queue_t * ibqp, sysinterval_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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

A byte value from the queue.

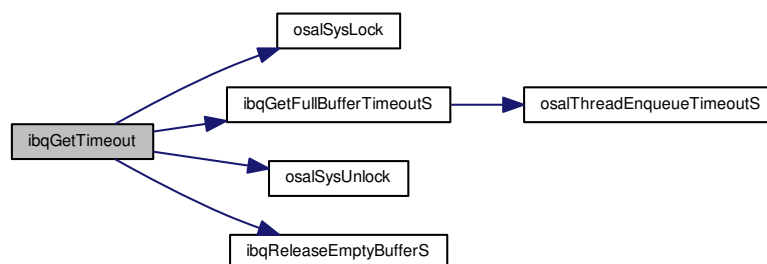
Return values

<code>MSG_TIMEOUT</code>	if the specified time expired.
<code>MSG_RESET</code>	if the queue has been reset or has been put in suspended 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:



7.13.4.10 `size_t ibqReadTimeout (input_buffers_queue_t * ibqp, uint8_t * bp, size_t n, sysinterval_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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The number of bytes effectively transferred.

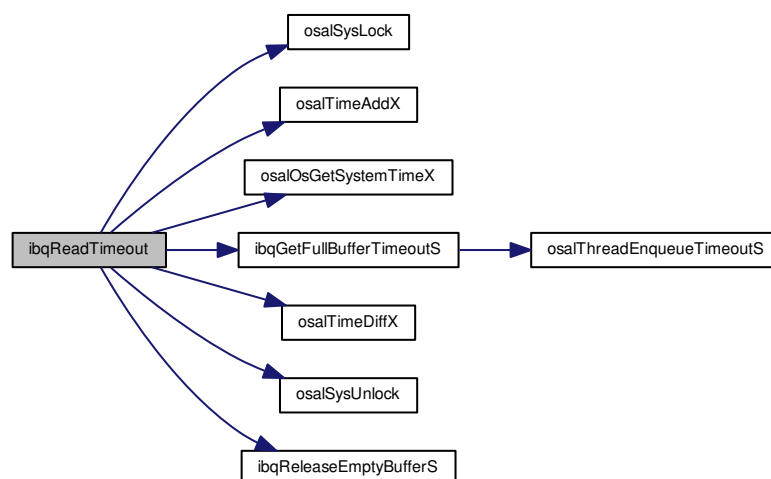
Return values

0	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:



7.13.4.11 void obqObjectInit (output_buffers_queue_t * obqp, bool suspended, uint8_t * bp, size_t size, size_t n, bqnotify_t onfy, void * link)

Initializes an output buffers queue object.

Parameters

out	<i>obqp</i>	pointer to the output_buffers_queue_t object
in	<i>suspended</i>	initial state of the queue
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>onfy</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:



7.13.4.12 void obqReset(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 output_buffers_queue_t 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:



7.13.4.13 `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.

7.13.4.14 `void obqReleaseEmptyBufferI (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:



7.13.4.15 msg_t obqGetEmptyBufferTimeout (output_buffers_queue_t * obqp, sysinterval_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>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

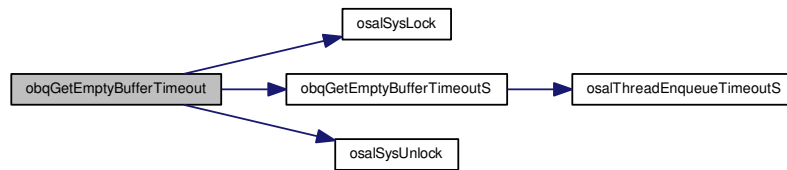
Return values

<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 or has been put in suspended 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:



7.13.4.16 msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t * obqp, sysinterval_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>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

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 or has been put in suspended state.

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:



7.13.4.17 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.

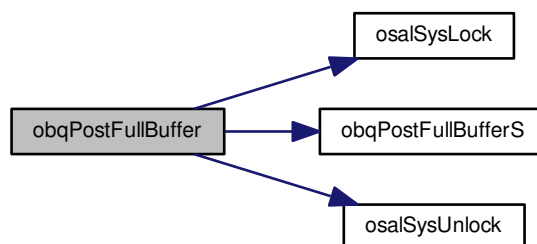
Parameters

in	<i>obqp</i>	pointer to the output_buffers_queue_t 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:



7.13.4.18 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 <code>output_buffers_queue_t</code> 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.

7.13.4.19 `msg_t obqPutTimeout(output_buffers_queue_t * obqp, uint8_t b, sysinterval_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 <code>output_buffers_queue_t</code> 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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

A byte value from the queue.

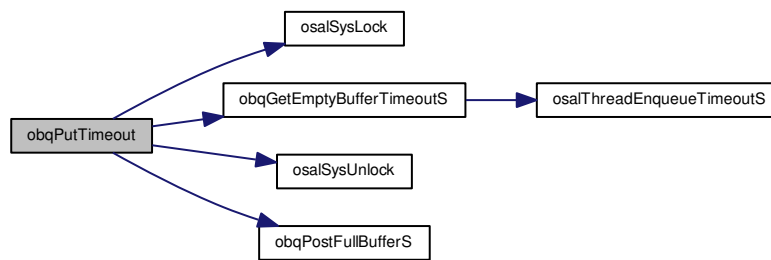
Return values

<code>MSG_TIMEOUT</code>	if the specified time expired.
<code>MSG_RESET</code>	if the queue has been reset or has been put in suspended 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:



7.13.4.20 `size_t obqWriteTimeout (output_buffers_queue_t * obqp, const uint8_t * bp, size_t n, sysinterval_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	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The number of bytes effectively transferred.

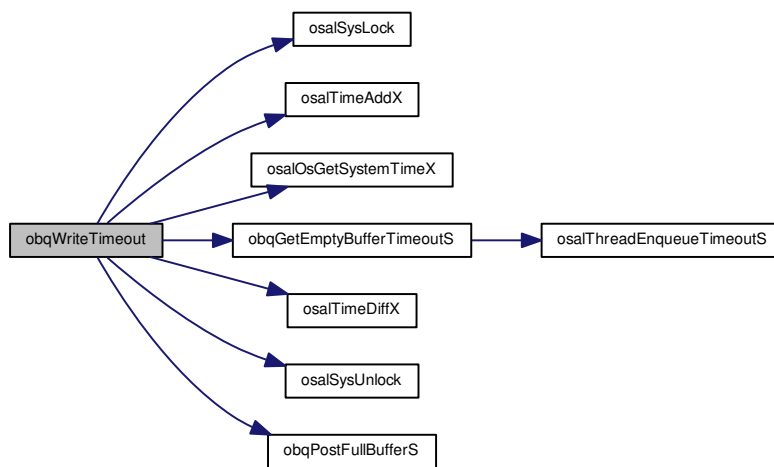
Return values

0	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:



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

Returns

The operation status.

Return values

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

7.13.4.22 `void obqFlush(output_buffers_queue_t * obqp)`

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

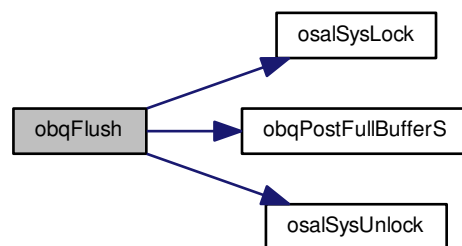
Parameters

in	<i>obqp</i>	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:



7.14 Abstract I/O Channel

7.14.1 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.

Default control operation codes.

- `#define CHN_CTL_INVALID 0 /** @brief Invalid operation code. */`
- `#define CHN_CTL_NOP 1 /** @brief Does nothing. */`
- `#define CHN_CTL_TX_WAIT 2 /** @brief Wait for TX completion. */`

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.
- `#define chnControl(ip, operation, arg) ((ip)->vmt->ctl(ip, operation, arg))`
Control operation on a channel.

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.

7.14.2 Macro Definition Documentation

7.14.2.1 #define _base_channel_methods

Value:

```
_base_sequential_stream_methods
/* Channel put method with timeout specification.*/
msg_t (*putt)(void *instance, uint8_t b, sysinterval_t time);
/* Channel get method with timeout specification.*/
msg_t (*gett)(void *instance, sysinterval_t time);
/* Channel write method with timeout specification.*/
size_t (*writet)(void *instance, const uint8_t *bp,
                size_t n, sysinterval_t time);
/* Channel read method with timeout specification.*/
size_t (*readt)(void *instance, uint8_t *bp, size_t n,
               sysinterval_t time);
\
/* Channel control method.*/
msg_t (*ctl)(void *instance, unsigned int operation, void *arg);
```

`BaseChannel` specific methods.

7.14.2.2 #define _base_channel_data _base_sequential_stream_data

`BaseChannel` specific data.

Note

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

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

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.

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

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.

7.14.2.5 `#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 BaseChannel 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.

7.14.2.6 `#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 BaseChannel 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"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

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.

7.14.2.7 `#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 BaseChannel 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.

7.14.2.8 `#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 BaseChannel 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"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

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.

7.14.2.9 `#define chnControl(ip, operation, arg) ((ip)->vmt->ctl(ip, operation, arg))`

Control operation on a channel.

Parameters

in	<i>ip</i>	pointer to a BaseChannel or derived class
in	<i>operation</i>	control operation code
in, out	<i>arg</i>	operation argument

Returns

The control operation status.

Return values

<i>MSG_OK</i>	in case of success.
<i>MSG_TIMEOUT</i>	in case of operation timeout.
<i>MSG_RESET</i>	in case of operation reset.

Function Class:

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

7.14.2.10 `#define CHN_NO_ERROR (eventflags_t)0`

No pending conditions.

7.14.2.11 `#define CHN_CONNECTED (eventflags_t)1`

Connection happened.

7.14.2.12 `#define CHN_DISCONNECTED (eventflags_t)2`

Disconnection happened.

7.14.2.13 `#define CHN_INPUT_AVAILABLE (eventflags_t)4`

Data available in the input queue.

7.14.2.14 `#define CHN_OUTPUT_EMPTY (eventflags_t)8`

Output queue empty.

7.14.2.15 `#define CHN_TRANSMISSION_END (eventflags_t)16`

Transmission end.

7.14.2.16 `#define _base_asynchronous_channel_methods _base_channel_methods \`

[BaseAsynchronousChannel](#) specific methods.

7.14.2.17 `#define _base_asynchronous_channel_data`

Value:

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

[BaseAsynchronousChannel](#) specific data.

7.14.2.18 `#define chnGetEventSource(ip) (&((ip)->event))`

Returns the I/O condition event source.

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

Parameters

in	<i>ip</i>	pointer to a BaseAsynchronousChannel 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.

7.14.2.19 #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	<i>ip</i>	pointer to a BaseAsynchronousChannel or derived class
in	<i>flags</i>	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.

7.15 Abstract Files

7.15.1 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.

7.15.2 Macro Definition Documentation

7.15.2.1 #define FILE_OK STM_OK

No error return code.

7.15.2.2 #define FILE_ERROR STM_TIMEOUT

Error code from the file stream methods.

7.15.2.3 #define FILE_EOF STM_RESET

End-of-file condition for file get/put methods.

7.15.2.4 #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.

7.15.2.5 #define _file_stream_data _base_sequential_stream_data

[FileStream](#) specific data.

Note

It is empty because [FileStream](#) is only an interface without implementation.

7.15.2.6 #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 FileStream 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

<code>FILE_ERROR</code>	operation failed.
-------------------------	-------------------

Function Class:

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

7.15.2.7 #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 FileStream 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

<code>FILE_ERROR</code>	operation failed.
-------------------------	-------------------

Function Class:

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

7.15.2.8 #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 FileStream 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.

7.15.2.9 #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 FileStream 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.

7.15.2.10 #define fileStreamClose(ip) ((ip)->vmt->close(ip))

File Stream close.

The function closes a file stream.

Parameters

in	<i>ip</i>	pointer to a FileStream 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.

7.15.2.11 `#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 FileStream 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.

7.15.2.12 `#define fileStreamGetSize(ip) ((ip)->vmt->getsize(ip))`

Returns the current file size.

Parameters

in	<i>ip</i>	pointer to a FileStream or derived class
----	-----------	--

Returns

The file size.

Return values

<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.

7.15.2.13 #define fileStreamGetPosition(*ip*) ((*ip*)->vmt->getposition(*ip*))

Returns the current file pointer position.

Parameters

in	<i>ip</i>	pointer to a FileStream or derived class
----	-----------	--

Returns

The current position inside the file.

Return values

<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.

7.15.2.14 #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 FileStream 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.

7.15.3 Typedef Documentation

7.15.3.1 typedef uint32_t fileoffset_t

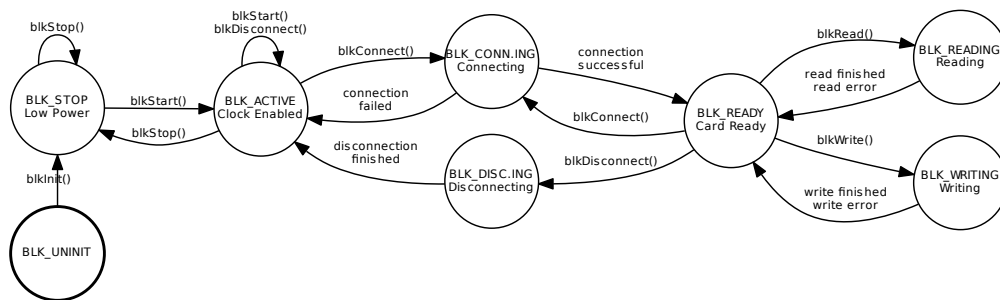
File offset type.

7.16 Abstract I/O Block Device

7.16.1 Detailed Description

7.16.2 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

7.16.3 Macro Definition Documentation

7.16.3.1 #define _base_block_device_methods

Value:

```
_base_object_methods
/* 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.

7.16.3.2 #define _base_block_device_data

Value:

```
_base_object_data
/* Driver state.*/
blkstate_t      state;
```

`BaseBlockDevice` specific data.

7.16.3.3 `#define blkGetDriverState(ip) ((ip)->state)`

Returns the driver state.

Note

Can be called in ISR context.

Parameters

in	<i>ip</i>	pointer to a BaseBlockDevice or derived class
----	-----------	---

Returns

The driver state.

Function Class:

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

7.16.3.4 `#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	<i>ip</i>	pointer to a BaseBlockDevice 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.

7.16.3.5 #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 <code>BaseBlockDevice</code> 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.

7.16.3.6 #define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))

Returns the media write protection status.

Parameters

in	<i>ip</i>	pointer to a <code>BaseBlockDevice</code> 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.

7.16.3.7 #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 BaseBlockDevice 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.

7.16.3.8 `#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 BaseBlockDevice 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.

7.16.3.9 `#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 BaseBlockDevice or derived class
in	<i>startblk</i>	first block to read

Parameters

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.

7.16.3.10 `#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 BaseBlockDevice 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.

7.16.3.11 `#define blkSync(ip) ((ip)->vmt->sync(ip))`

Ensures write synchronization.

Parameters

in	<i>ip</i>	pointer to a BaseBlockDevice 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.

7.16.3.12 `#define blkGetInfo(ip, bdip) ((ip)->vmt->get_info(ip, bdip))`

Returns a media information structure.

Parameters

in	<i>ip</i>	pointer to a BaseBlockDevice or derived class
out	<i>bdip</i>	pointer to a BlockDeviceInfo 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.

7.16.4 Enumeration Type Documentation**7.16.4.1 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.

7.17 Base Object

7.17.1 Detailed Description

HAL uses concepts of Object Oriented Programming even if it is written in C. Things like simple inheritance, multiple inheritance and interfaces are used through the system. This module defines a "base object" that is the ancestor of all classes in the system.

Macros

- `#define _base_object_methods`
BaseObject specific methods.
- `#define _base_object_data`
BaseObject specific data.

Macro Functions (BaseObject)

- `#define objGetInstance(type, ip) (type)(((size_t)(ip)) - (ip)->vmt->instance_offset)`
Returns the instance pointer starting from an interface pointer.

Data Structures

- struct `BaseObjectVMT`
BaseObject virtual methods table.
- struct `BaseObject`
Base stream class.

7.17.2 Macro Definition Documentation

7.17.2.1 `#define _base_object_methods`

Value:

```
/* Instance offset, used for multiple inheritance, normally zero. It
   represents the offset between the current object and the container
   object*/
size_t instance_offset;
```

`BaseObject` specific methods.

Note

This object defines no methods.

7.17.2.2 `#define _base_object_data`

`BaseObject` specific data.

Note

This object defines no data.

7.17.2.3 `#define objGetInstance(type, ip) (type)((size_t)(ip) - (ip)->vmt->instance_offset)`

Returns the instance pointer starting from an interface pointer.

Parameters

in	<i>type</i>	the type of the instance pointer, it is used for casting
in	<i>ip</i>	the interface pointer

Returns

A pointer to the object implementing the interface

7.18 Abstract Peripheral Interfaces

HAL Abstract Peripheral Interfaces.

7.18.1 Detailed Description

HAL Abstract Peripheral Interfaces.

Modules

- [Generic Accelerometer Interface](#)
HAL Generic Accelerometer Interface.
- [Generic Barometer Interface](#)
HAL Generic Barometer Interface.
- [Generic Compass Interface](#)
HAL Generic Compass Interface.
- [Generic Flash Driver Interface](#)
HAL Generic Flash Driver Interface.
- [Generic Gyroscope Interface](#)
HAL Generic Gyroscope Interface.
- [Generic Hygrometer Interface](#)
HAL Generic Hygrometer Interface.
- [JESD216 Compliant Flash Driver Interface](#)
HAL JESD216 Compliant Flash Driver Interface.
- [Generic Sensor Interface](#)
HAL Generic Sensor Interface.
- [Generic Thermometer Interface](#)
HAL Generic Thermometer Interface.

7.19 I/O Bytes Queues

7.19.1 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_TIMEOUT`
Queue empty.
- `#define Q_FULL MSG_TIMEOUT`
Queue full,.

Macro Functions

- `#define qSizeX(qp)`
Returns the queue's buffer size.
- `#define qSpacel(qp) ((qp)->q_counter)`
Queue space.
- `#define qGetLink(qp) ((qp)->q_link)`
Returns the queue application-defined link.
- `#define iqGetFullI(iqp) qSpacel(iqp)`
Returns the filled space into an input queue.
- `#define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpacel(iqp))`
Returns the empty space into an input queue.
- `#define iqlsEmptyI(iqp) ((bool)(qSpacel(iqp) == 0U))`
Evaluates to `true` if the specified input queue is empty.
- `#define iqlsFullI(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) - qSpacel(oqp))`
Returns the filled space into an output queue.
- `#define oqGetEmptyI(oqp) qSpacel(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)(qSpacel(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

- `static size_t iq_read (input_queue_t *iqp, uint8_t *bp, size_t n)`
Non-blocking input queue read.
- `static size_t oq_write (output_queue_t *oqp, const uint8_t *bp, size_t n)`
Non-blocking output queue write.
- `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 iqGetl (input_queue_t *iqp)`
Input queue non-blocking read.
- `msg_t iqGetTimeout (input_queue_t *iqp, sysinterval_t timeout)`
Input queue read with timeout.
- `size_t iqReadl (input_queue_t *iqp, uint8_t *bp, size_t n)`
Input queue non-blocking read.
- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, sysinterval_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 oqPutl (output_queue_t *oqp, uint8_t b)`
Output queue non-blocking write.
- `msg_t oqPutTimeout (output_queue_t *oqp, uint8_t b, sysinterval_t timeout)`

Output queue write with timeout.

- `msg_t oqGetl (output_queue_t *oqp)`

Output queue read.

- `size_t oqWritel (output_queue_t *oqp, const uint8_t *bp, size_t n)`

Output queue non-blocking write.

- `size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, sysinterval_t timeout)`

Output queue write with timeout.

7.19.2 Macro Definition Documentation

7.19.2.1 #define Q_OK MSG_OK

Operation successful.

7.19.2.2 #define Q_TIMEOUT MSG_TIMEOUT

Timeout condition.

7.19.2.3 #define Q_RESET MSG_RESET

Queue has been reset.

7.19.2.4 #define Q_EMPTY MSG_TIMEOUT

Queue empty.

7.19.2.5 #define Q_FULL MSG_TIMEOUT

Queue full,.

7.19.2.6 #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.

7.19.2.7 #define qSpacel(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	<i>qp</i>	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.

7.19.2.8 #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.

7.19.2.9 #define iqGetFullI(iqp) qSpacel(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

0	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.

7.19.2.10 `#define iqGetEmptyl(iqp) (qSizeX(iqp) - qSpacel(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

0	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.

7.19.2.11 `#define iqlsEmptyl(iqp) ((bool)(qSpacel(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.

7.19.2.12 #define iqlsFull(iqp)

Value:

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/
((bool) (((iqp)->q_wrptr == (iqp)->q_rdptr) && ((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.

7.19.2.13 #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>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.

7.19.2.14 #define oqGetFull(oqp) (qSizeX(oqp) - qSpacel(oqp))

Returns the filled space into an output queue.

Parameters

in	oqp	pointer to an output_queue_t structure
----	-----	--

Returns

The number of full bytes in the queue.

Return values

0	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.

7.19.2.15 #define oqGetEmpty(oqp) qSpacel(oqp)

Returns the empty space into an output queue.

Parameters

in	oqp	pointer to an output_queue_t structure
----	-----	--

Returns

The number of empty bytes in the queue.

Return values

0	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.

7.19.2.16 #define oqlsEmpty(oqp)

Value:

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/ \
((bool) (((oqp)->q_wrptr == (oqp)->q_rdptr) && ((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.

7.19.2.17 `#define oqlsFull(oqp) ((bool)(qSpace1(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.

7.19.2.18 `#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>MSG_OK</i>	if the operation succeeded.
<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.

7.19.3 Typedef Documentation**7.19.3.1 typedef struct io_queue io_queue_t**

Type of a generic I/O queue structure.

7.19.3.2 typedef void(* qnotify_t)(io_queue_t *qp)

Queue notification callback type.

Parameters

in	<i>qp</i>	the queue pointer
----	-----------	-------------------

7.19.3.3 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.

7.19.3.4 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.

7.19.4 Function Documentation**7.19.4.1 static size_t iq_read (input_queue_t * iqp, uint8_t * bp, size_t n) [static]**

Non-blocking input queue read.

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

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

Returns

The number of bytes effectively transferred.

Function Class:

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

7.19.4.2 `static size_t oq_write (output_queue_t * oqp, const uint8_t * bp, size_t n)` [static]

Non-blocking output queue write.

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

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

Returns

The number of bytes effectively transferred.

Function Class:

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

7.19.4.3 `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.

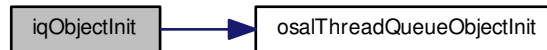
Parameters

out	<i>iqp</i>	pointer to an <code>input_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>infy</i>	pointer to a callback function that is invoked when data is read from the queue. The value can be NULL.
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:



7.19.4.4 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 `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.

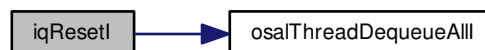
Parameters

in	<i>iqp</i>	pointer to an <code>input_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:



7.19.4.5 msg_t iqPutI (input_queue_t * iqp, uint8_t b)

Input queue write.

A byte value is written into the low end of an input queue. The operation completes immediately.

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.

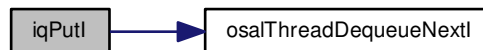
Return values

<i>MSG_OK</i>	if the operation has been completed with success.
<i>MSG_TIMEOUT</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.

Here is the call graph for this function:

**7.19.4.6 msg_t iqGetl (input_queue_t * iqp)**

Input queue non-blocking read.

This function reads a byte value from an input queue. The operation completes immediately.

Note

The callback is invoked after removing a character from 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>MSG_TIMEOUT</i>	if the queue is empty.
<i>MSG_RESET</i>	if the queue has been reset.

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.

7.19.4.7 msg_t iqGetTimeout (input_queue_t * iqp, sysinterval_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 after removing a character from the queue.

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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

A byte value from the queue.

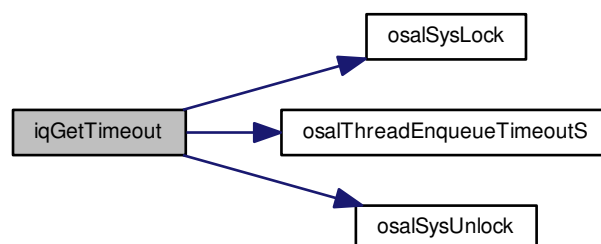
Return values

<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:



7.19.4.8 `size_t iqReadI (input_queue_t * iqp, uint8_t * bp, size_t n)`

Input queue non-blocking read.

The function reads data from an input queue into a buffer. The operation completes immediately.

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

Returns

The number of bytes effectively transferred.

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:



7.19.4.9 `size_t iqReadTimeout (input_queue_t * iqp, uint8_t * bp, size_t n, sysinterval_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 after removing each character from the queue.

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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

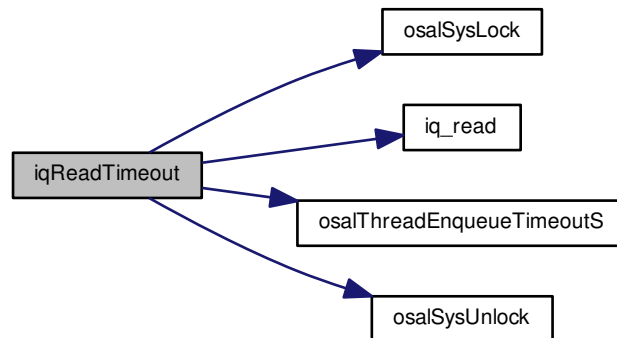
Returns

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:



7.19.4.10 `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>onfy</i>	pointer to a callback function that is invoked when data is written to the queue. The value can be NULL.
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:



7.19.4.11 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 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>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:



7.19.4.12 msg_t oqPutI (output_queue_t * oqp, uint8_t b)

Output queue non-blocking write.

This function writes a byte value to an output queue. The operation completes immediately.

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>MSG_OK</i>	if the operation succeeded.
<i>MSG_TIMEOUT</i>	if the queue is full.
<i>MSG_RESET</i>	if the queue has been reset.

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.

7.19.4.13 msg_t oqPutTimeout (output_queue_t * oqp, uint8_t b, sysinterval_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 putting the character into 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
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

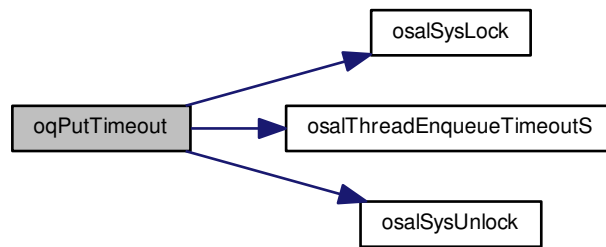
Return values

<i>MSG_OK</i>	if the operation succeeded.
<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:



7.19.4.14 `msg_t oqGetl (output_queue_t * oqp)`

Output queue read.

A byte value is read from the low end of an output queue. The operation completes immediately.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

Returns

The byte value from the queue.

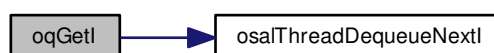
Return values

<code>MSG_TIMEOUT</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.

Here is the call graph for this function:



7.19.4.15 `size_t oqWritel (output_queue_t * oqp, const uint8_t * bp, size_t n)`

Output queue non-blocking write.

The function writes data from a buffer to an output queue. The operation completes immediately.

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

Returns

The number of bytes effectively transferred.

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:



7.19.4.16 `size_t oqWriteTimeout (output_queue_t * oqp, const uint8_t * bp, size_t n, sysinterval_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 putting each character into the queue.

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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

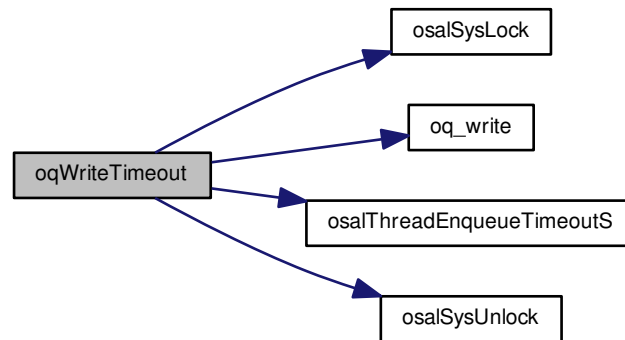
Returns

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:



7.20 Abstract Streams

7.20.1 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_base_object_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.

7.20.2 Macro Definition Documentation

7.20.2.1 `#define _base_sequential_stream_methods`

Value:

```
_base_object_methods
/* 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.

7.20.2.2 #define _base_sequential_stream_data_base_object_data

[BaseSequentialStream](#) specific data.

Note

It is empty because [BaseSequentialStream](#) is only an interface without implementation.

7.20.2.3 #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	<i>ip</i>	pointer to a BaseSequentialStream 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.

Function Class:

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

7.20.2.4 #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	<i>ip</i>	pointer to a BaseSequentialStream 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.

Function Class:

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

7.20.2.5 `#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 BaseChannel 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.

7.20.2.6 `#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 BaseChannel 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.

7.21 Generic Hygrometer Interface

HAL Generic Hygrometer Interface.

7.21.1 Detailed Description

HAL Generic Hygrometer Interface.

Macros

- `#define _base_hygrometer_methods_alone`
BaseHygrometer specific methods.
- `#define _base_hygrometer_methods`
BaseHygrometer specific methods with inherited ones.
- `#define _base_hygrometer_data _base_sensor_data`
BaseHygrometer specific data.

Macro Functions (BaseHygrometer)

- `#define hygrometerGetChannelsNumber(ip) (ip)->vmt->get_channels_number(ip)`
Hygrometer get channels number.
- `#define hygrometerReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)`
Hygrometer read raw data.
- `#define hygrometerReadCooked(ip, dp) (ip)->vmt->read_cooked(ip, dp)`
Hygrometer read cooked data.
- `#define hygrometerSetBias(ip, bp) (ip)->vmt->set_bias(ip, bp)`
Updates hygrometer bias data from received buffer.
- `#define hygrometerResetBias(ip) (ip)->vmt->reset_bias(ip)`
Reset hygrometer bias data restoring it to zero.
- `#define hygrometerSetSensitivity(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)`
Updates hygrometer sensitivity data from received buffer.
- `#define hygrometerResetSensitivity(ip) (ip)->vmt->reset_sensitivity(ip)`
Reset hygrometer sensitivity data restoring it to its typical value.

Data Structures

- `struct BaseHygrometerVMT`
BaseHygrometer virtual methods table.
- `struct BaseHygrometer`
Base hygrometer class.

7.21.2 Macro Definition Documentation

7.21.2.1 `#define _base_hygrometer_methods_alone`

Value:

```

/* Invoke the set bias procedure.*/
msg_t (*set_bias)(void *instance, float biases[]);
/* Remove bias stored data.*/
msg_t (*reset_bias)(void *instance);
/* Invoke the set sensitivity procedure.*/
msg_t (*set_sensitivity)(void *instance, float sensitivities[]);
/* Restore sensitivity stored data to default.*/
msg_t (*reset_sensitivity)(void *instance);

```

[BaseHygrometer](#) specific methods.

7.21.2.2 #define _base_hygrometer_methods

Value:

```

_base_sensor_methods
_base_hygrometer_methods_alone

```

[BaseHygrometer](#) specific methods with inherited ones.

7.21.2.3 #define _base_hygrometer_data _base_sensor_data

[BaseHygrometer](#) specific data.

7.21.2.4 #define hygrometerGetChannelsNumber(ip) (ip)->vmt->get_channels_number(ip)

Hygrometer get channels number.

Parameters

in	<i>ip</i>	pointer to a BaseHygrometer class.
----	-----------	--

Returns

The number of channels of the [BaseHygrometer](#)

Function Class:

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

7.21.2.5 #define hygrometerReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)

Hygrometer read raw data.

Parameters

in	<i>ip</i>	pointer to a BaseHygrometer class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.21.2.6 #define hygrometerReadCooked(*ip*, *dp*)(ip)->vmt->read_cooked(ip, dp)

Hygrometer read cooked data.

Parameters

in	<i>ip</i>	pointer to a BaseHygrometer class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.21.2.7 #define hygrometerSetBias(*ip*, *bp*)(ip)->vmt->set_bias(ip, bp)

Updates hygrometer bias data from received buffer.

Note

The bias buffer must have the same length of the the hygrometer channels number.

Parameters

in	<i>ip</i>	pointer to a BaseHygrometer class.
in	<i>bp</i>	pointer to a buffer of bias values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.21.2.8 `#define hygrometerResetBias(ip) (ip)->vmt->reset_bias(ip)`

Reset hygrometer bias data restoring it to zero.

Parameters

in	<i>ip</i>	pointer to a BaseHygrometer class.
----	-----------	--

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.21.2.9 `#define hygrometerSetSensitivity(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)`

Updates hygrometer sensitivity data from received buffer.

Note

The sensitivity buffer must have the same length of the the hygrometer channels number.

Parameters

in	<i>ip</i>	pointer to a BaseHygrometer class.
in	<i>sp</i>	pointer to a buffer of sensitivity values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.21.2.10 `#define hygrometerResetSensitivity(ip) (ip)->vmt->reset_sensitivity(ip)`

Reset hygrometer sensitivity data restoring it to its typical value.

Parameters

in	<i>ip</i>	pointer to a BaseHygrometer class.
----	-----------	--

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.22 I2C Driver

Generic I2C Driver.

7.22.1 Detailed Description

Generic I2C Driver.

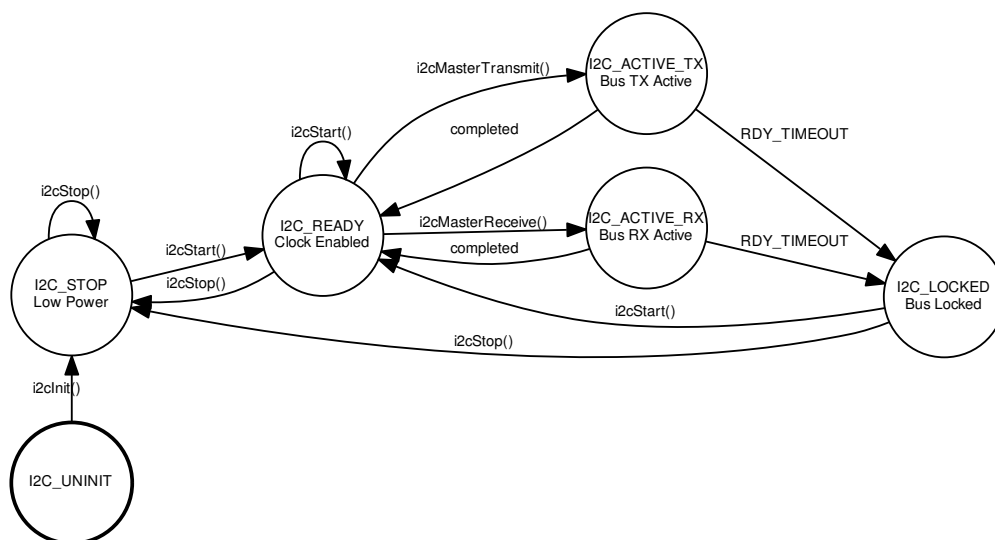
This module implements a generic I2C (Inter-Integrated Circuit) driver.

Precondition

In order to use the I2C driver the `HAL_USE_I2C` option must be enabled in `halconf.h`.

7.22.2 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)`

- *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.
- `#define i2c_Ild_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` `i2cMasterTransmitTimeout` (`I2CDriver` *i2cp, `i2caddr_t` addr, const `uint8_t` *txbuf, `size_t` txbytes, `uint8_t` *rxbuf, `size_t` rxbytes, `sysinterval_t` timeout)
Sends data via the I2C bus.
- `msg_t` `i2cMasterReceiveTimeout` (`I2CDriver` *i2cp, `i2caddr_t` addr, `uint8_t` *rxbuf, `size_t` rxbytes, `sysinterval_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` *rxbuf, `size_t` rxbytes, `sysinterval_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` *rxbuf, `size_t` rxbytes, `sysinterval_t` timeout)
Transmits data via the I2C bus as master.

Enumerations

Variables

- `I2CDriver` `I2CD1`
I2C1 driver identifier.

7.22.3 Macro Definition Documentation

7.22.3.1 #define I2C_NO_ERROR 0x00

No error.

7.22.3.2 #define I2C_BUS_ERROR 0x01

Bus Error.

7.22.3.3 #define I2C_ARBITRATION_LOST 0x02

Arbitration Lost.

7.22.3.4 #define I2C_ACK_FAILURE 0x04

Acknowledge Failure.

7.22.3.5 #define I2C_OVERRUN 0x08

Overrun/Underrun.

7.22.3.6 #define I2C_PEC_ERROR 0x10

PEC Error in reception.

7.22.3.7 #define I2C_TIMEOUT 0x20

Hardware timeout.

7.22.3.8 #define I2C_SMB_ALERT 0x40

SMBus Alert.

7.22.3.9 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.22.3.10 #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	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

7.22.3.11 #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 I2CDriver object
----	-------------	---

Function Class:

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

7.22.3.12 `#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.

7.22.3.13 `#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.

7.22.3.14 `#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`.

7.22.3.15 `#define i2c_lld_get_errors(i2cp) ((i2cp)->errors)`

Get errors from I2C driver.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

7.22.4 Typedef Documentation

7.22.4.1 typedef uint16_t i2caddr_t

Type representing an I2C address.

7.22.4.2 typedef uint32_t i2cflags_t

Type of I2C Driver condition flags.

7.22.4.3 typedef struct I2CDriver I2CDriver

Type of a structure representing an I2C driver.

7.22.5 Enumeration Type Documentation

7.22.5.1 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.

7.22.6 Function Documentation

7.22.6.1 void i2cInit (void)

I2C Driver initialization.

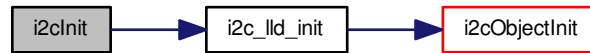
Note

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:



7.22.6.2 void i2cObjectInit (I2CDriver * i2cp)

Initializes the standard part of a [I2CDriver](#) structure.

Parameters

out	<i>i2cp</i>	pointer to the I2CDriver 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:



7.22.6.3 void i2cStart (I2CDriver * i2cp, const I2CConfig * config)

Configures and activates the I2C peripheral.

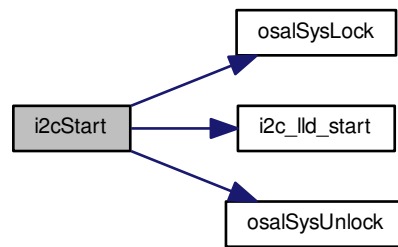
Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
in	<i>config</i>	pointer to the I2CConfig 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:



7.22.6.4 void i2cStop (I2CDriver * i2cp)

Deactivates the I2C peripheral.

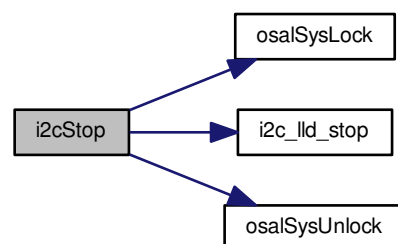
Parameters

in	<i>i2cp</i>	pointer to the I2CDriver 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:



7.22.6.5 i2cflags_t i2cGetErrors (I2CDriver * i2cp)

Returns the errors mask associated to the previous operation.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver 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.

7.2.2.6.6 `msg_t i2cMasterTransmitTimeout (I2CDriver * i2cp, i2caddr_t addr, const uint8_t * txbuf, size_t txbytes, uint8_t * rxbuf, size_t rxbytes, sysinterval_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 I2CDriver 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"> • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

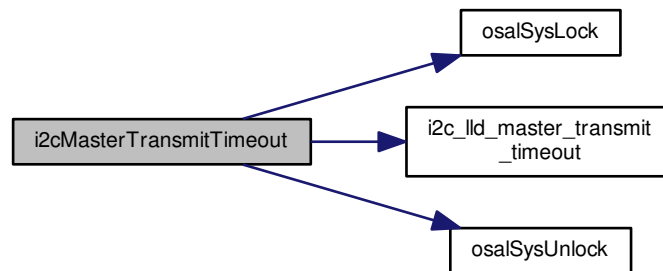
Return values

<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 i2cGetErrors() .
<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:



7.22.6.7 `msg_t i2cMasterReceiveTimeout (I2CDriver * i2cp, i2caddr_t addr, uint8_t * rxbuf, size_t rxbytes, sysinterval_t timeout)`

Receives data from the I2C bus.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
in	<i>addr</i>	slave device address (7 bits) without R/W bit
out	<i>rxbuf</i>	pointer to 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"> • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

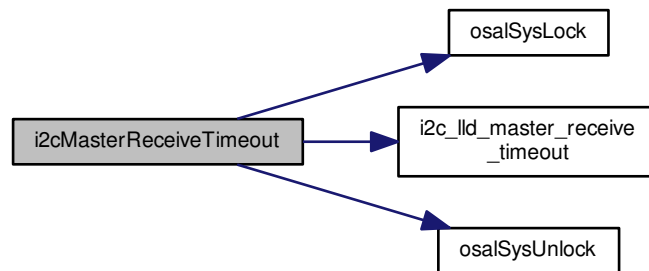
Return values

<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 i2cGetErrors() .
<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:



7.22.6.8 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	<i>i2cp</i>	pointer to the I2CDriver 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:



7.22.6.9 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 I2CDriver 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:

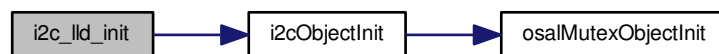
**7.22.6.10 void i2c_lld_init (void)**

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:

**7.22.6.11 void i2c_lld_start (I2CDriver * i2cp)**

Configures and activates the I2C peripheral.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

7.22.6.12 void i2c_lld_stop (I2CDriver * i2cp)

Deactivates the I2C peripheral.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

7.22.6.13 msg_t i2c_lld_master_receive_timeout (I2CDriver * i2cp, i2caddr_t addr, uint8_t * rxbuf, size_t rxbytes, sysinterval_t timeout)

Receives data via the I2C bus as master.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver 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"> • <i>TIME_INFINITE</i> no timeout.

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 i2cGetErrors() .
<i>MSG_TIMEOUT</i>	if a timeout occurred before operation end. After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.

Function Class:

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

7.22.6.14 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, sysinterval_t timeout)

Transmits data via the I2C bus as master.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver 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>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none">• <i>TIME_INFINITE</i> no timeout.

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 i2cGetErrors() .
<i>MSG_TIMEOUT</i>	if a timeout occurred before operation end. After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.

Function Class:

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

7.22.7 Variable Documentation

7.22.7.1 I2CDriver I2CD1

I2C1 driver identifier.

7.23 I2S Driver

Generic I2S Driver.

7.23.1 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`.

7.23.2 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_llc_init` (void)
Low level I2S driver initialization.
- void `i2s_llc_start` (I2SDriver *i2sp)
Configures and activates the I2S peripheral.
- void `i2s_llc_stop` (I2SDriver *i2sp)
Deactivates the I2S peripheral.
- void `i2s_llc_start_exchange` (I2SDriver *i2sp)
Starts a I2S data exchange.
- void `i2s_llc_stop_exchange` (I2SDriver *i2sp)
Stops the ongoing data exchange.

Enumerations

Variables

- `I2SDriver I2SD1`
I2S2 driver identifier.

7.23.3 Macro Definition Documentation

7.23.3.1 #define i2sStartExchange(i2sp)

Value:

```
{
    i2s_llc_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.

7.23.3.2 #define i2sStopExchange(i2sp)

Value:

```
{
    i2s_ll_d_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 I2SDriver 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.

7.23.3.3 #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 I2CDriver object
----	-------------	---

Function Class:

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

7.23.3.4 #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 I2CDriver object
----	-------------	---

Function Class:

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

7.23.3.5 #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`.

7.23.4 Typedef Documentation

7.23.4.1 typedef struct I2SDriver I2SDriver

Type of a structure representing an I2S driver.

7.23.4.2 typedef void(* i2scallback_t) (I2SDriver *i2sp, size_t offset, size_t n)

I2S notification callback type.

Parameters

in	<i>i2sp</i>	pointer to the I2SDriver object
in	<i>offset</i>	offset in buffers of the data to read/write
in	<i>n</i>	number of samples to read/write

7.23.5 Enumeration Type Documentation

7.23.5.1 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.

7.23.6 Function Documentation

7.23.6.1 void i2sInit (void)

I2S Driver initialization.

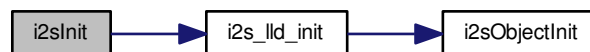
Note

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:



7.23.6.2 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.

7.23.6.3 void i2sStart (I2SDriver * *i2sp*, const I2SConfig * *config*)

Configures and activates the I2S peripheral.

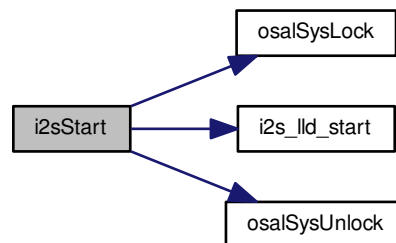
Parameters

in	<i>i2sp</i>	pointer to the I2SDriver object
in	<i>config</i>	pointer to the I2SConfig 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:

**7.23.6.4 void i2sStop (I2SDriver * *i2sp*)**

Deactivates the I2S peripheral.

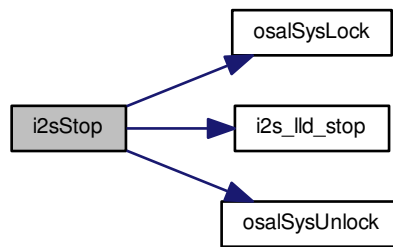
Parameters

in	<i>i2sp</i>	pointer to the I2SDriver 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:



7.23.6.5 void i2sStartExchange (I2SDriver * i2sp)

Starts a I2S data exchange.

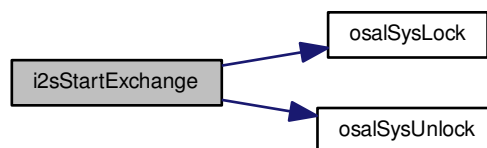
Parameters

in	<i>i2sp</i>	pointer to the I2SDriver 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:



7.23.6.6 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.

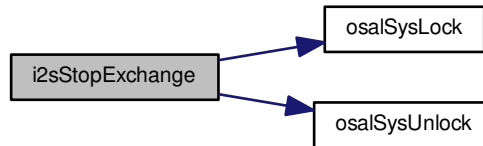
Parameters

in	<i>i2sp</i>	pointer to the I2SDriver 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:

**7.23.6.7 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:

**7.23.6.8 void i2s_lld_start (I2SDriver * i2sp)**

Configures and activates the I2S peripheral.

Parameters

in	<i>i2sp</i>	pointer to the I2SDriver object
----	-------------	---

Function Class:

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

7.23.6.9 void i2s_lld_stop (I2SDriver * i2sp)

Deactivates the I2S peripheral.

Parameters

in	<i>i2sp</i>	pointer to the I2SDriver object
----	-------------	---

Function Class:

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

7.23.6.10 void i2s_lld_start_exchange (I2SDriver * i2sp)

Starts a I2S data exchange.

Parameters

in	<i>i2sp</i>	pointer to the I2SDriver object
----	-------------	---

Function Class:

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

7.23.6.11 void i2s_lld_stop_exchange (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	<i>i2sp</i>	pointer to the I2SDriver object
----	-------------	---

Function Class:

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

7.23.7 Variable Documentation**7.23.7.1 I2SDriver I2SD1**

I2S2 driver identifier.

7.24 ICU Driver

Generic ICU Driver.

7.24.1 Detailed Description

Generic ICU Driver.

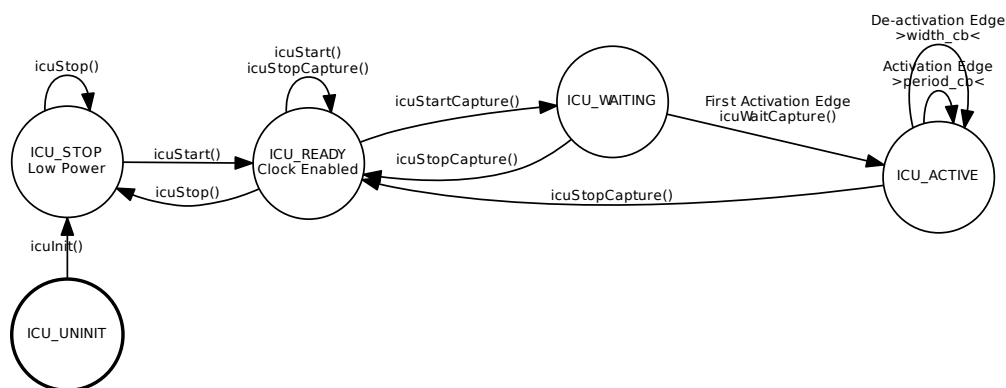
This module implements a generic ICU (Input Capture Unit) driver. The purpose of the driver is to measure period and duty cycle of an input digital signal (PWM input).

Precondition

In order to use the ICU driver the `HAL_USE_ICU` option must be enabled in `halconf.h`.

7.24.2 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).



7.24.3 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 icuStartCaptureI(icup)`
Starts the input capture.
- `#define icuStopCaptureI(icup)`
Stops the input capture.
- `#define icuEnableNotificationsI(icup) icu_lld_enable_notifications(icup)`
Enables notifications.
- `#define icuDisableNotificationsI(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 [iculnit](#) (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

Variables

- `ICUDriver ICUD1`
ICUD1 driver identifier.

7.24.4 Macro Definition Documentation

7.24.4.1 #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.

7.24.4.2 #define icuStopCapture(icup)

Value:

```
do {
    icu_lld_stop_capture(icup);
    (icup)->state = ICU_READY;
} while (false)
```

Stops 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.

7.24.4.3 #define icuEnableNotifications(*icup*) icu_ild_enable_notifications(icup)

Enables notifications.

Precondition

The ICU unit must have been activated using `icuStart()` and the capture started using `icuStartCapture()`.

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:

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

7.24.4.4 #define icuDisableNotifications(*icup*) icu_ild_disable_notifications(icup)

Disables notifications.

Precondition

The ICU unit must have been activated using `icuStart()` and the capture started using `icuStartCapture()`.

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:

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 #define icuAreNotificationsEnabledX(*icup*) icu_ild_are_notifications_enabled(icup)

Check on notifications status.

Parameters

in	<i>icup</i>	pointer to the <code>ICUDriver</code> 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.

7.24.4.6 #define icuGetWidthX(*icup*) icu_ild_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 ICUDriver object
----	-------------	---

Returns

The number of ticks.

Function Class:

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

7.24.4.7 #define icuGetPeriodX(*icup*) icu_ild_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 ICUDriver object
----	-------------	---

Returns

The number of ticks.

Function Class:

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

7.24.4.8 #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 ICUDriver object
----	-------------	---

Function Class:

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

7.24.4.9 #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 ICUDriver object
----	-------------	---

Function Class:

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

7.24.4.10 #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 ICUDriver object
----	-------------	---

Function Class:

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

7.24.4.11 #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`.

7.24.4.12 #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 ICUDriver object
----	-------------	---

Returns

The number of ticks.

Function Class:

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

7.24.4.13 #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 ICUDriver object
----	-------------	---

Returns

The number of ticks.

Function Class:

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

7.24.4.14 `#define icu_lld_are_notifications_enabled(icup) false`

Check on notifications status.

Parameters

in	<i>icup</i>	pointer to the ICUDriver 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.

7.24.5 Typedef Documentation

7.24.5.1 `typedef struct ICUDriver ICUDriver`

Type of a structure representing an ICU driver.

7.24.5.2 `typedef void(* icucallback_t)(ICUDriver *icup)`

ICU notification callback type.

Parameters

in	<i>icup</i>	pointer to a ICUDriver object
----	-------------	---

7.24.5.3 `typedef uint32_t icufreq_t`

ICU frequency type.

7.24.5.4 `typedef uint32_t icucnt_t`

ICU counter type.

7.24.6 Enumeration Type Documentation

7.24.6.1 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.

7.24.6.2 enum icumode_t

ICU driver mode.

Enumerator

ICU_INPUT_ACTIVE_HIGH Trigger on rising edge.

ICU_INPUT_ACTIVE_LOW Trigger on falling edge.

7.24.7 Function Documentation

7.24.7.1 void icuInit (void)

ICU Driver initialization.

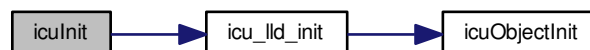
Note

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:



7.24.7.2 void icuObjectInit (ICUDriver * icup)

Initializes the standard part of a `ICUDriver` structure.

Parameters

out	<i>icup</i>	pointer to the ICUDriver object
-----	-------------	---

Function Class:

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

7.24.7.3 void icuStart (ICUDriver * *icup*, const ICUConfig * *config*)

Configures and activates the ICU peripheral.

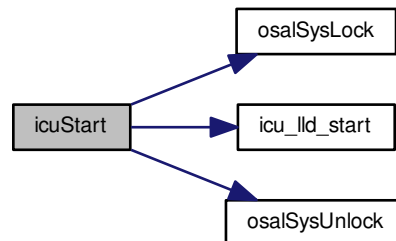
Parameters

in	<i>icup</i>	pointer to the ICUDriver object
in	<i>config</i>	pointer to the ICUConfig 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:

**7.24.7.4 void icuStop (ICUDriver * *icup*)**

Deactivates the ICU peripheral.

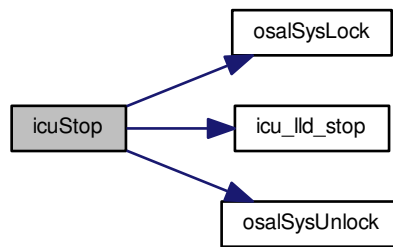
Parameters

in	<i>icup</i>	pointer to the ICUDriver 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:



7.24.7.5 void icuStartCapture (ICUDriver * icup)

Starts the input capture.

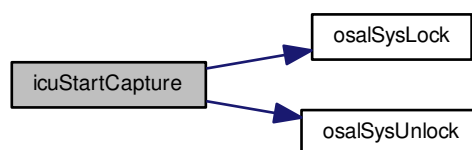
Parameters

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:



7.24.7.6 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 <code>ICUDriver</code> object
----	-------------	--

Returns

The capture status.

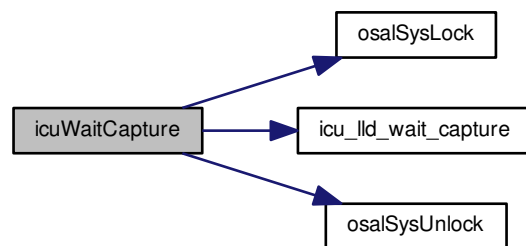
Return values

<i>false</i>	if the capture is successful.
<i>true</i>	if a timer overflow occurred.

Function Class:

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:

**7.24.7.7 void icuStopCapture (ICUDriver * icup)**

Stops the input capture.

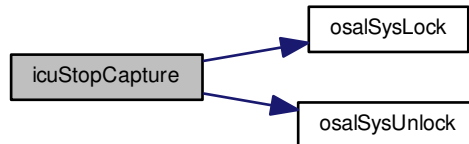
Parameters

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:

**7.24.7.8 void icuEnableNotifications (ICUDriver * icup)**

Enables notifications.

Precondition

The ICU unit must have been activated using `icuStart()` and the capture started using `icuStartCapture()`.

Note

If the notification is already enabled then the call has no effect.

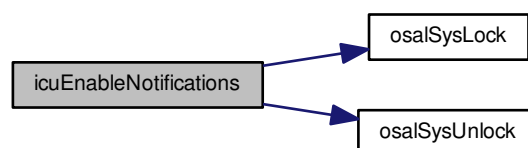
Parameters

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:



7.24.7.9 void icuDisableNotifications (ICUDriver * icup)

Disables notifications.

Precondition

The ICU unit must have been activated using `icuStart()` and the capture started using `icuStartCapture()`.

Note

If the notification is already disabled then the call has no effect.

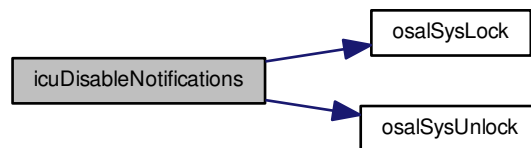
Parameters

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:



7.24.7.10 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:



7.24.7.11 void icu_lld_start (ICUDriver * icup)

Configures and activates the ICU peripheral.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.24.7.12 void icu_lld_stop (ICUDriver * icup)

Deactivates the ICU peripheral.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.24.7.13 void icu_lld_start_capture (ICUDriver * icup)

Starts the input capture.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.24.7.14 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 ICUDriver 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.

7.24.7.15 void icu_lld_stop_capture (ICUDriver * icup)

Stops the input capture.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.24.7.16 void icu_lld_enable_notifications (ICUDriver * icup)

Enables notifications.

Precondition

The ICU unit must have been activated using [icuStart\(\)](#) and the capture started using [icuStartCapture\(\)](#).

Note

If the notification is already enabled then the call has no effect.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.24.7.17 void icu_lld_disable_notifications (ICUDriver * icup)

Disables notifications.

Precondition

The ICU unit must have been activated using `icuStart()` and the capture started using `icuStartCapture()`.

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.

7.24.8 Variable Documentation**7.24.8.1 ICUDriver ICUD1**

ICUD1 driver identifier.

Note

The driver ICUD1 allocates the complex timer TIM1 when enabled.

7.25 JESD216 Compliant Flash Driver Interface

HAL JESD216 Compliant Flash Driver Interface.

7.25.1 Detailed Description

HAL JESD216 Compliant Flash Driver Interface.

Macros

- `#define _jesd216_flash_methods_alone`
JESD215Flash specific methods.
- `#define _jesd216_flash_methods`
JESD215Flash specific methods with inherited ones.
- `#define _jesd216_flash_data_base_flash_data`
JESD215Flash specific data.

Common command codes

- `#define JESD216_CMD_READ_ID 0x9FU`
- `#define JESD216_CMD_READ 0x03U`
- `#define JESD216_CMD_WRITE_ENABLE 0x06U`
- `#define JESD216_CMD_WRITE_DISABLE 0x04U`
- `#define JESD216_CMD_READ_STATUS_REGISTER 0x05U`
- `#define JESD216_CMD_WRITE_STATUS_REGISTER 0x01U`
- `#define JESD216_CMD_PAGE_PROGRAM 0x02U`
- `#define JESD216_CMD_ERASE_4K 0x20U`
- `#define JESD216_CMD_ERASE_BULK 0xC7U`
- `#define JESD216_CMD_PROGRAM_ERASE_RESUME 0x7AU`
- `#define JESD216_CMD_PROGRAM_ERASE_SUSPEND 0x75U`
- `#define JESD216_CMD_READ_OTP_ARRAY 0x4BU`
- `#define JESD216_CMD_PROGRAM_OTP_ARRAY 0x42U`

Command options

- `#define JESD216_CMD_EXTENDED_ADDRESSING 0x80000000U`

Bus interface.

- `#define JESD216_BUS_MODE_SPI 0U`
- `#define JESD216_BUS_MODE_QSPI1L 1U`
- `#define JESD216_BUS_MODE_QSPI2L 2U`
- `#define JESD216_BUS_MODE_QSPI4L 4U`

Configuration options

- `#define JESD216_BUS_MODE JESD216_BUS_MODE_QSPI4L`
Physical transport interface.
- `#define JESD216_SHARED_BUS TRUE`
Shared bus switch.

Data Structures

- struct [JESD215FlashVMT](#)
JESD215Flash virtual methods table.
- struct [JESD215Flash](#)
Base flash class.

7.25.2 Macro Definition Documentation

7.25.2.1 #define JESD216_BUS_MODE JESD216_BUS_MODE_QSPI4L

Physical transport interface.

7.25.2.2 #define JESD216_SHARED_BUS TRUE

Shared bus switch.

If set to `TRUE` the device acquires bus ownership on each transaction.

Note

Requires `SPI_USE_MUTUAL_EXCLUSION` or `SPI_USE_MUTUAL_EXCLUSION`.

7.25.2.3 #define _jesd216_flash_methods_alone

Value:

```
/* Read SFDP.*/
flash_error_t (*read_sfdp)(void *instance,
    flash_offset_t offset,
    size_t n,
    uint8_t *rp);
```

[JESD215Flash](#) specific methods.

7.25.2.4 #define _jesd216_flash_methods

Value:

```
_base_flash_methods
_jesd216_flash_methods_alone
```

[JESD215Flash](#) specific methods with inherited ones.

7.25.2.5 #define _jesd216_flash_data _base_flash_data

[JESD215Flash](#) specific data.

7.26 MAC Driver

Generic MAC Driver.

7.26.1 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)`
Enables the zero-copy API.
- `#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, [sysinterval_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, [sysinterval_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

Variables

- `MACDriver ETHD1`
MAC1 driver identifier.

7.26.2 Macro Definition Documentation

7.26.2.1 #define MAC_USE_ZERO_COPY FALSE

Enables an event sources for incoming packets.

7.26.2.2 #define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

7.26.2.3 #define macGetReceiveEventSource(macp) (&(macp)->rdevent)

Enables the zero-copy API.

Parameters

in	<i>macp</i>	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.

7.26.2.4 #define macWriteTransmitDescriptor(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)

Writes to a transmit descriptor's stream.

Parameters

in	<i>tdp</i>	pointer to a <code>MACTransmitDescriptor</code> 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:

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

7.26.2.5 `#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 MACReceiveDescriptor 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.

7.26.2.6 `#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 MACTransmitDescriptor 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.

7.26.2.7 `#define macGetNextReceiveBuffer(rdp, sizep) mac_ild_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

in	<i>rdp</i>	pointer to a MACReceiveDescriptor 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

<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.

7.26.2.8 `#define MAC_SUPPORTS_ZERO_COPY TRUE`

This implementation supports the zero-copy mode API.

7.26.2.9 `#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`.

7.26.3 Typedef Documentation

7.26.3.1 `typedef struct MACDriver MACDriver`

Type of a structure representing a MAC driver.

7.26.4 Enumeration Type Documentation

7.26.4.1 `enum macstate_t`

Driver state machine possible states.

Enumerator

MAC_UNINIT Not initialized.

MAC_STOP Stopped.

MAC_ACTIVE Active.

7.26.5 Function Documentation

7.26.5.1 void macInit (void)

MAC Driver initialization.

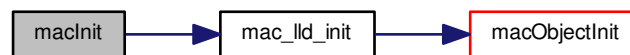
Note

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:



7.26.5.2 void macObjectInit (MACDriver * macp)

Initialize the standard part of a `MACDriver` structure.

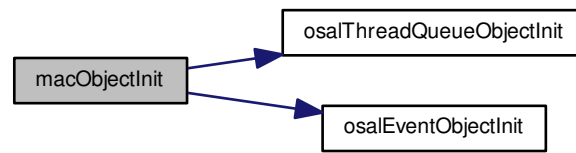
Parameters

out	<i>macp</i>	pointer to the <code>MACDriver</code> 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:



7.26.5.3 void macStart (MACDriver * macp, const MACConfig * config)

Configures and activates the MAC peripheral.

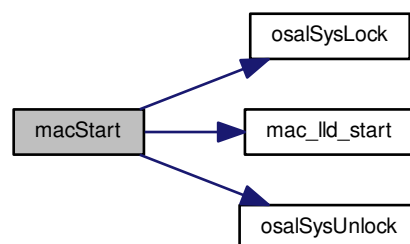
Parameters

in	<i>macp</i>	pointer to the MACDriver object
in	<i>config</i>	pointer to the MACConfig 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:



7.26.5.4 void macStop (MACDriver * macp)

Deactivates the MAC peripheral.

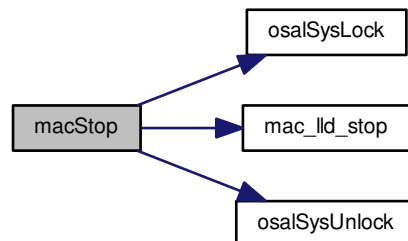
Parameters

in	<i>macp</i>	pointer to the MACDriver 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:



7.26.5.5 `msg_t macWaitTransmitDescriptor (MACDriver * macp, MACTransmitDescriptor * tdp, sysinterval_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 MACDriver object
out	<i>tdp</i>	pointer to a MACTransmitDescriptor 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"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

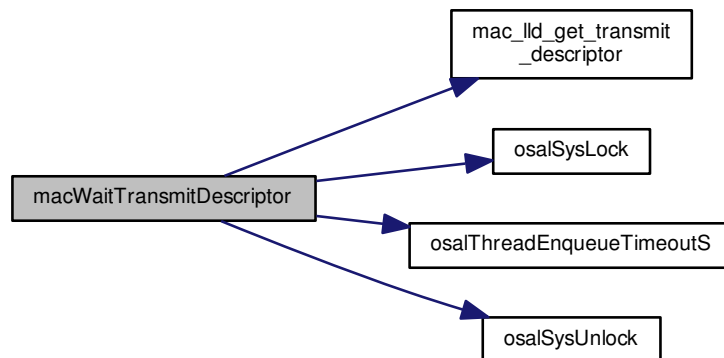
Return values

<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:



7.26.5.6 void macReleaseTransmitDescriptor (MACTransmitDescriptor * tdp)

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

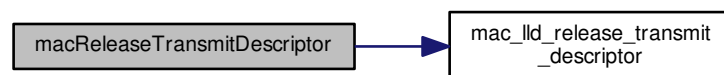
Parameters

in	<i>tdp</i>	the pointer to the MACTransmitDescriptor 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:



7.26.5.7 msg_t macWaitReceiveDescriptor (MACDriver * macp, MACReceiveDescriptor * rdp, sysinterval_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 MACDriver object
out	<i>rdp</i>	pointer to a MACReceiveDescriptor 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"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

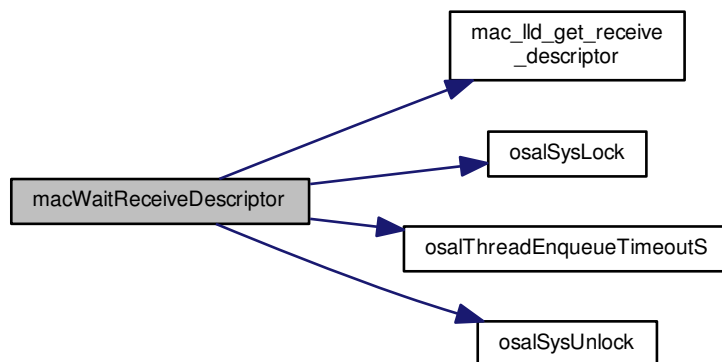
Return values

<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:



7.26.5.8 void macReleaseReceiveDescriptor ([MACReceiveDescriptor](#) * rdp)

Releases a receive descriptor.

The descriptor and its buffer are made available for more incoming frames.

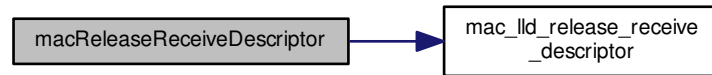
Parameters

in	<i>rdp</i>	the pointer to the MACReceiveDescriptor 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:



7.26.5.9 bool macPollLinkStatus (MACDriver * macp)

Updates and returns the link status.

Parameters

in	macp	pointer to the MACDriver 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:



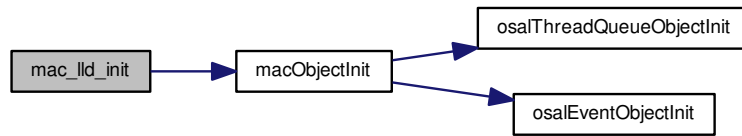
7.26.5.10 void mac_llid_init (void)

Low level MAC initialization.

Function Class:

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

Here is the call graph for this function:



7.26.5.11 void mac_lld_start (MACDriver * macp)

Configures and activates the MAC peripheral.

Parameters

in	macp	pointer to the MACDriver object
----	------	---

Function Class:

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

7.26.5.12 void mac_lld_stop (MACDriver * macp)

Deactivates the MAC peripheral.

Parameters

in	macp	pointer to the MACDriver object
----	------	---

Function Class:

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

7.26.5.13 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	macp	pointer to the MACDriver object
out	tdp	pointer to a MACTransmitDescriptor 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.

7.26.5.14 void mac_llid_release_transmit_descriptor (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 MACTransmitDescriptor structure
----	------------	--

Function Class:

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

7.26.5.15 msg_t mac_llid_get_receive_descriptor (MACDriver * macp, MACReceiveDescriptor * rdp)

Returns a receive descriptor.

Parameters

in	<i>macp</i>	pointer to the MACDriver object
out	<i>rdp</i>	pointer to a MACReceiveDescriptor 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.

7.26.5.16 void mac_llid_release_receive_descriptor (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 MACReceiveDescriptor structure
----	------------	---

Function Class:

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

7.26.5.17 bool mac_llt_poll_link_status (MACDriver * macp)

Updates and returns the link status.

Parameters

in	<i>macp</i>	pointer to the MACDriver 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.

7.26.5.18 size_t mac_llt_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 MACTransmitDescriptor 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.

7.26.5.19 `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 MACReceiveDescriptor 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.

7.26.5.20 `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 MACTransmitDescriptor 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.

7.26.5.21 `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 MACReceiveDescriptor 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

<i>NULL</i>	if the buffer chain has been entirely scanned.
-------------	--

Function Class:

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

7.26.6 Variable Documentation**7.26.6.1 MACDriver ETHD1**

MAC1 driver identifier.

7.27 HAL

Hardware Abstraction Layer.

7.27.1 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.

7.27.2 HAL Device Drivers Architecture

The HAL contains several kind of modules:

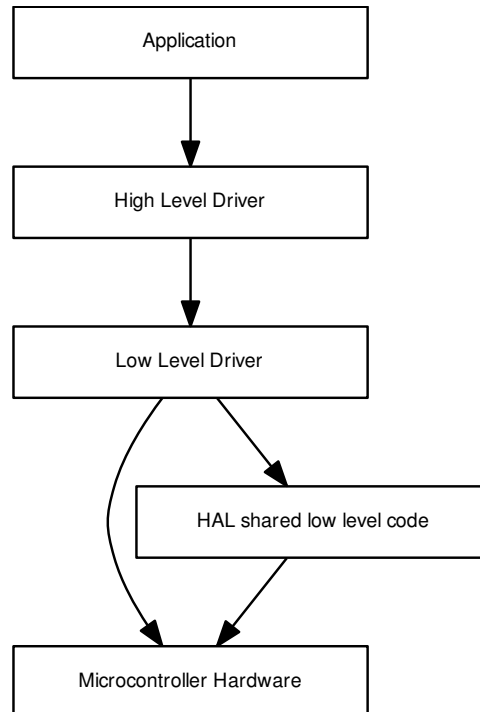
- Normal Device Drivers
- Complex Device Drivers
- Interfaces
- Inner Code

7.27.3 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.

7.27.3.1 Diagram



7.27.4 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.

7.27.5 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.

7.27.6 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 and Classes](#)

HAL Interfaces and Classes.

- [Inner Code](#)

HAL Inner Code.

- [Support Code](#)

HAL Support Code.

- [OSAL](#)

Operating System Abstraction Layer.

7.28 Configuration

HAL Configuration.

7.28.1 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 describe in the various platforms reference manuals.

Macros

- `#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_CRY TRUE`
Enables the cryptographic subsystem.
- `#define HAL_USE_DAC TRUE`
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_QSPI TRUE`
Enables the QSPI 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`
- Enables the USB subsystem.*

 - #define `HAL_USE_WDG` `TRUE`
- Enables the WDG subsystem.*

 - #define `PAL_USE_CALLBACKS` `FALSE`
- Enables synchronous APIs.*

 - #define `PAL_USE_WAIT` `FALSE`
- Enables synchronous APIs.*

 - #define `ADC_USE_WAIT` `TRUE`
- Enables synchronous APIs.*

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

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

 - #define `CAN_ENFORCE_USE_CALLBACKS` `FALSE`
- Enforces the driver to use direct callbacks rather than OSAL events.*

 - #define `HAL_CRY_USE_FALLBACK` `FALSE`
- Enables the SW fall-back of the cryptographic driver.*

 - #define `HAL_CRY_ENFORCE_FALLBACK` `FALSE`
- Makes the driver forcibly use the fall-back implementations.*

 - #define `DAC_USE_WAIT` `TRUE`
- Enables synchronous APIs.*

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

 - #define `I2C_USE_MUTUAL_EXCLUSION` `TRUE`
- Enables the mutual exclusion APIs on the I2C bus.*

 - #define `MAC_USE_ZERO_COPY` `TRUE`
- Enables the zero-copy API.*

 - #define `MAC_USE_EVENTS` `TRUE`
- Enables an event sources for incoming packets.*

 - #define `MMC_NICE_WAITING` `TRUE`
- Delays insertions.*

 - #define `QSPI_USE_WAIT` `TRUE`
- Enables synchronous APIs.*

 - #define `QSPI_USE_MUTUAL_EXCLUSION` `TRUE`
- Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.*

 - #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.*

 - #define `SDC_INIT_OCR_V20` `0x50FF8000U`
- OCR initialization constant for V20 cards.*

 - #define `SDC_INIT_OCR` `0x80100000U`
- OCR initialization constant for non-V20 cards.*

 - #define `SERIAL_DEFAULT_BITRATE` `38400`
- Default bit rate.*

- `#define SERIAL_BUFFERS_SIZE 16`
Serial buffers size.
- `#define SERIAL_USB_BUFFERS_SIZE 256`
Serial over USB buffers size.
- `#define SERIAL_USB_BUFFERS_NUMBER 2`
Serial over USB number of buffers.
- `#define SPI_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define SPI_USE_CIRCULAR FALSE`
Enables circular transfers APIs.
- `#define SPI_USE_MUTUAL_EXCLUSION TRUE`
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.
- `#define SPI_SELECT_MODE SPI_SELECT_MODE_PAD`
Handling method for SPI CS line.
- `#define UART_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define UART_USE_MUTUAL_EXCLUSION TRUE`
Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.
- `#define USB_USE_WAIT TRUE`
Enables synchronous APIs.

7.28.2 Macro Definition Documentation

7.28.2.1 `#define HAL_USE_PAL TRUE`

Enables the PAL subsystem.

7.28.2.2 `#define HAL_USE_ADC TRUE`

Enables the ADC subsystem.

7.28.2.3 `#define HAL_USE_CAN TRUE`

Enables the CAN subsystem.

7.28.2.4 `#define HAL_USE_CRY TRUE`

Enables the cryptographic subsystem.

7.28.2.5 `#define HAL_USE_DAC TRUE`

Enables the DAC subsystem.

7.28.2.6 `#define HAL_USE_EXT TRUE`

Enables the EXT subsystem.

7.28.2.7 `#define HAL_USE_GPT TRUE`

Enables the GPT subsystem.

7.28.2.8 `#define HAL_USE_I2C TRUE`

Enables the I2C subsystem.

7.28.2.9 `#define HAL_USE_I2S TRUE`

Enables the I2S subsystem.

7.28.2.10 `#define HAL_USE_ICU TRUE`

Enables the ICU subsystem.

7.28.2.11 `#define HAL_USE_MAC TRUE`

Enables the MAC subsystem.

7.28.2.12 `#define HAL_USE_MMC_SPI TRUE`

Enables the MMC_SPI subsystem.

7.28.2.13 `#define HAL_USE_PWM TRUE`

Enables the PWM subsystem.

7.28.2.14 `#define HAL_USE_QSPI TRUE`

Enables the QSPI subsystem.

7.28.2.15 `#define HAL_USE_RTC TRUE`

Enables the RTC subsystem.

7.28.2.16 `#define HAL_USE_SDC TRUE`

Enables the SDC subsystem.

7.28.2.17 `#define HAL_USE_SERIAL TRUE`

Enables the SERIAL subsystem.

7.28.2.18 `#define HAL_USE_SERIAL_USB TRUE`

Enables the SERIAL over USB subsystem.

7.28.2.19 `#define HAL_USE_SPI TRUE`

Enables the SPI subsystem.

7.28.2.20 #define HAL_USE_UART TRUE

Enables the UART subsystem.

7.28.2.21 #define HAL_USE_USB TRUE

Enables the USB subsystem.

7.28.2.22 #define HAL_USE_WDG TRUE

Enables the WDG subsystem.

7.28.2.23 #define PAL_USE_CALLBACKS FALSE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.28.2.24 #define PAL_USE_WAIT FALSE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.28.2.25 #define ADC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.28.2.26 #define ADC_USE_MUTUAL_EXCLUSION TRUE

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

Note

Disabling this option saves both code and data space.

7.28.2.27 #define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

7.28.2.28 #define CAN_ENFORCE_USE_CALLBACKS FALSE

Enforces the driver to use direct callbacks rather than OSAL events.

7.28.2.29 #define HAL_CRY_USE_FALLBACK FALSE

Enables the SW fall-back of the cryptographic driver.

When enabled, this option, activates a fall-back software implementation for algorithms not supported by the underlying hardware.

Note

Fall-back implementations may not be present for all algorithms.

7.28.2.30 #define HAL_CRY_ENFORCE_FALLBACK FALSE

Makes the driver forcibly use the fall-back implementations.

7.28.2.31 #define DAC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.28.2.32 #define DAC_USE_MUTUAL_EXCLUSION TRUE

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

Note

Disabling this option saves both code and data space.

7.28.2.33 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.28.2.34 #define MAC_USE_ZERO_COPY TRUE

Enables the zero-copy API.

7.28.2.35 #define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

7.28.2.36 #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.

7.28.2.37 `#define QSPI_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.28.2.38 `#define QSPI_USE_MUTUAL_EXCLUSION TRUE`

Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.28.2.39 `#define SDC_INIT_RETRY 100`

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.28.2.40 `#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`.

7.28.2.41 `#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.

7.28.2.42 `#define SDC_INIT_OCR_V20 0x50FF8000U`

OCR initialization constant for V20 cards.

7.28.2.43 `#define SDC_INIT_OCR 0x80100000U`

OCR initialization constant for non-V20 cards.

7.28.2.44 `#define SERIAL_DEFAULT_BITRATE 38400`

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.28.2.45 #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.

7.28.2.46 #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.

7.28.2.47 #define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.28.2.48 #define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.28.2.49 #define SPI_USE_CIRCULAR FALSE

Enables circular transfers APIs.

Note

Disabling this option saves both code and data space.

7.28.2.50 #define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.28.2.51 `#define SPI_SELECT_MODE SPI_SELECT_MODE_PAD`

Handling method for SPI CS line.

Note

Disabling this option saves both code and data space.

7.28.2.52 `#define UART_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.28.2.53 `#define UART_USE_MUTUAL_EXCLUSION TRUE`

Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.28.2.54 `#define USB_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.29 Normal Drivers

HAL Normal Drivers.

7.29.1 Detailed Description

HAL Normal Drivers.

Modules

- [ADC Driver](#)
Generic ADC Driver.
- [CAN Driver](#)
Generic CAN Driver.
- [Crypto Driver](#)
Generic Crypto 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.
- [QSPI Driver](#)
Generic QSPI 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.

7.30 Complex Drivers

HAL Complex Drivers.

7.30.1 Detailed Description

HAL Complex Drivers.

Modules

- [Managed Flash Storage Driver](#)
Managed Flash Storage Driver.
- [MMC over SPI Driver](#)
Generic MMC driver.
- [Serial over USB Driver](#)
Serial over USB Driver.

7.31 Interfaces and Classes

HAL Interfaces and Classes.

7.31.1 Detailed Description

HAL Interfaces and Classes.

Modules

- [Abstract I/O Channel](#)
- [Abstract Files](#)
- [Abstract I/O Block Device](#)
- [Base Object](#)
- [Abstract Peripheral Interfaces](#)
 - HAL Abstract Peripheral Interfaces.*
- [Abstract Streams](#)

7.32 Inner Code

HAL Inner Code.

7.32.1 Detailed Description

HAL Inner Code.

Modules

- [I/O Buffers Queues](#)
- [I/O Bytes Queues](#)
- [MMC/SD Block Device](#)

7.33 Support Code

HAL Support Code.

7.33.1 Detailed Description

HAL Support Code.

Modules

- [MII/RMII Header](#)
MII/RMII Support Header.
- [USB CDC Header](#)
USB CDC Support Header.

7.34 OSAL

Operating System Abstraction Layer.

7.34.1 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 thin 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_TIMEOUT (msg_t)-1`
- `#define MSG_RESET (msg_t)-2`

Special time constants

- `#define TIME_IMMEDIATE ((sysinterval_t)0)`
- `#define TIME_INFINITE ((sysinterval_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_S2I`(secs) ((`sysinterval_t`)((uint32_t)(secs) * (uint32_t)`OSAL_ST_FREQUENCY`))
Seconds to system ticks.
- #define `OSAL_MS2I`(msecs)
Milliseconds to system ticks.
- #define `OSAL_US2I`(usecs)
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(secs) osalThreadSleep(OSAL_S2I(secs))`
Delays the invoking thread for the specified number of seconds.
- `#define osalThreadSleepMilliseconds(msecs) osalThreadSleep(OSAL_MS2I(msecs))`
Delays the invoking thread for the specified number of milliseconds.
- `#define osalThreadSleepMicroseconds(usecs) osalThreadSleep(OSAL_US2I(usecs))`
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 sysinterval_t`
Type of system time interval.
- `typedef uint32_t rtcnt_t`
Type of realtime counter.
- `typedef void * thread_reference_t`
Type of a thread reference.
- `typedef uint32_t eventflags_t`
Type of an event flags mask.
- `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 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 osalInit (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.*
- `sys_t osalOsGetSystemTimeX` (void)
 - Current system time.*
- void `osalThreadSleepS` (sysinterval_t time)
 - Suspends the invoking thread for the specified time.*
- void `osalThreadSleep` (sysinterval_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, sysinterval_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, sysinterval_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 `sys_t osalTimeAddX` (sys_t systime, sysinterval_t interval)
 - Adds an interval to a system time returning a system time.*
- static `sysinterval_t osalTimeDiffX` (sys_t start, sys_t end)
 - Subtracts two system times returning an interval.*

- static bool `osalTimeIsInRangeX` (`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 source 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.

7.34.2 Macro Definition Documentation

7.34.2.1 #define OSAL_ST_RESOLUTION 32

Size in bits of the `systick_t` type.

7.34.2.2 #define OSAL_ST_FREQUENCY 1000

Required systick frequency or resolution.

7.34.2.3 #define OSAL_ST_MODE OSAL_ST_MODE_PERIODIC

Systick mode required by the underlying OS.

7.34.2.4 #define OSAL_IRQ_PRIORITY_LEVELS 16U

Total priority levels.

Implementation not mandatory.

7.34.2.5 #define OSAL_IRQ_MAXIMUM_PRIORITY 0U

Highest IRQ priority for HAL drivers.

Implementation not mandatory.

7.34.2.6 #define OSAL_DBG_ENABLE_ASSERTS FALSE

Enables OSAL assertions.

7.34.2.7 #define OSAL_DBG_ENABLE_CHECKS FALSE

Enables OSAL functions parameters checks.

7.34.2.8 #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.

7.34.2.9 #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.

7.34.2.10 `#define osalDbgCheckClassl()`

I-Class state check.

Note

Implementation is optional.

7.34.2.11 `#define osalDbgCheckClassS()`

S-Class state check.

Note

Implementation is optional.

7.34.2.12 `#define OSAL_IRQ_IS_VALID_PRIORITY(n) (((n) >= OSAL_IRQ_MAXIMUM_PRIORITY) && ((n) < OSAL_IRQ_PRIORITY_LEVELS))`

Priority level verification macro.

7.34.2.13 `#define OSAL_IRQ_PROLOGUE()`

IRQ prologue code.

This macro must be inserted at the start of all IRQ handlers.

7.34.2.14 `#define OSAL_IRQ_EPILOGUE()`

IRQ epilogue code.

This macro must be inserted at the end of all IRQ handlers.

7.34.2.15 `#define OSAL_IRQ_HANDLER(id) void id(void)`

IRQ handler function declaration.

This macro hides the details of an ISR function declaration.

Parameters

<code>in</code>	<code>id</code>	a vector name as defined in <code>vectors.s</code>
-----------------	-----------------	--

7.34.2.16 `#define OSAL_S2l(secs) ((sysinterval_t)((uint32_t)(secs) * (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	secs	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.

7.34.2.17 #define OSAL_MS2I(msec)**Value:**

```
((sysinterval_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.

7.34.2.18 #define OSAL_US2I(usec)**Value:**

```
((sysinterval_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	<i>usecs</i>	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.

7.34.2.19 `#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.

7.34.2.20 `#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 >= 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.

7.34.2.21 `#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 >= 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.

7.34.2.22 `#define osalThreadSleepSeconds(secs) osalThreadSleep(OSAL_S2I(secs))`

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	<i>secs</i>	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.

7.34.2.23 `#define osalThreadSleepMilliseconds(msecs) osalThreadSleep(OSAL_MS2I(msecs))`

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	<i>msecs</i>	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.

7.34.2.24 `#define osalThreadSleepMicroseconds(usecs) osalThreadSleep(OSAL_US2I(usecs))`

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	<i>usecs</i>	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.

7.34.3 Typedef Documentation

7.34.3.1 `typedef uint32_t syssts_t`

Type of a system status word.

7.34.3.2 `typedef int32_t msg_t`

Type of a message.

7.34.3.3 `typedef uint32_t systime_t`

Type of system time counter.

7.34.3.4 `typedef uint32_t sysinterval_t`

Type of system time interval.

7.34.3.5 `typedef uint32_t rtcnt_t`

Type of realtime counter.

7.34.3.6 typedef void* thread_reference_t

Type of a thread reference.

7.34.3.7 typedef uint32_t eventflags_t

Type of an event flags mask.

7.34.3.8 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.

7.34.3.9 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.

7.34.3.10 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.

7.34.4 Function Documentation**7.34.4.1 void osallnit (void)**

OSAL module initialization.

Function Class:

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

7.34.4.2 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:



7.34.4.3 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.

7.34.4.4 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.

7.34.4.5 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.

7.34.4.6 `sys_time_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 `sys_time_t` size.

Returns

The system time in ticks.

Function Class:

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

7.34.4.7 `void osalThreadSleepS (sysinterval_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"> • <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification. • <code>TIME_IMMEDIATE</code> this value is not allowed.

Function Class:

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

7.34.4.8 `void osalThreadSleep (sysinterval_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"> • <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification. • <code>TIME_IMMEDIATE</code> this value is not allowed.

Function Class:

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

7.34.4.9 `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.

7.34.4.10 `msg_t osalThreadSuspendTimeoutS (thread_reference_t * trp, sysinterval_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"> • <i>TIME_INFINITE</i> the thread enters an infinite sleep state. • <i>TIME_IMMEDIATE</i> the thread is not enqueued and the function returns <i>MSG_TIMEOUT</i> as if a timeout occurred.

Returns

The wake up message.

Return values

<i>MSG_TIMEOUT</i>	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.

7.34.4.11 void osalThreadResume1 (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.

7.34.4.12 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.

7.34.4.13 msg_t osalThreadEnqueueTimeoutS (threads_queue_t * tqp, sysinterval_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>	the timeout in system ticks, the special values are handled as follow: <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> the thread enters an infinite sleep state. • <i>TIME_IMMEDIATE</i> the thread is not enqueued and the function returns MSG_TIMEOUT as if a timeout occurred.

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.

7.34.4.14 `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.

7.34.4.15 `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.

7.34.4.16 `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.

7.34.4.17 void osalEventBroadcastFlags (event_source_t * esp, eventflags_t flags)

Add flags to an event source object.

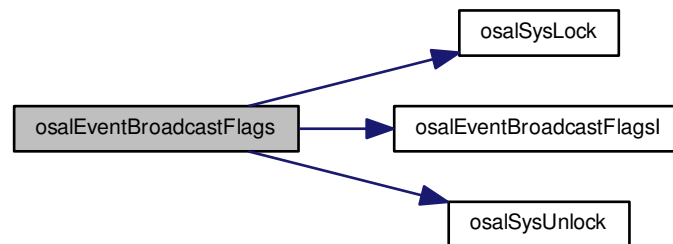
Parameters

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:



7.34.4.18 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.

7.34.4.19 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 mutex_t object
---------	-----------	-------------------------------

Function Class:

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

7.34.4.20 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 mutex_t object
---------	-----------	-------------------------------

Function Class:

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

7.34.4.21 static void osalSysDisable (void) [inline], [static]

Disables interrupts globally.

Function Class:

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

7.34.4.22 static void osalSysEnable (void) [inline], [static]

Enables interrupts globally.

Function Class:

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

7.34.4.23 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.

7.34.4.24 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.

7.34.4.25 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.

7.34.4.26 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.

7.34.4.27 `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.

7.34.4.28 `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.

7.34.4.29 `static systime_t osalTimeAddX (systime_t systime, sysinterval_t interval) [inline],[static]`

Adds an interval to a system time returning a system time.

Parameters

in	<i>systime</i>	base system time
in	<i>interval</i>	interval to be added

Returns

The new system time.

Function Class:

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

7.34.4.30 `static sysinterval_t osalTimeDiffX (systime_t start, systime_t end) [inline],[static]`

Subtracts two system times returning an interval.

Parameters

in	<i>start</i>	first system time
in	<i>end</i>	second system time

Returns

The interval representing the time difference.

Function Class:

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

7.34.4.31 `static bool osalTimeIsInRangeX (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.

7.34.4.32 `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.

7.34.4.33 `static void osalEventObjectInit (event_source_t* esp)` `[inline],[static]`

Initializes an event source object.

Parameters

out	<i>esp</i>	pointer to the event source object
-----	------------	------------------------------------

Function Class:

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

7.34.4.34 `static void osalMutexObjectInit (mutex_t* mp)` `[inline],[static]`

Initializes s `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.

7.34.5 Variable Documentation

7.34.5.1 `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.

7.34.5.2 `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.

7.35 Managed Flash Storage Driver

Managed Flash Storage Driver.

7.35.1 Detailed Description

Managed Flash Storage Driver.

This module implements a managed flash storage able to store a finite number of variable-size records. Records are retrieved by their index number.

The driver is automatically performs:

- Wear leveling.
- Auto repair after power loss.
- Garbage collection in order to remove erased data.

Macros

- `#define RET_ON_ERROR(err)`

Error check helper.

Configuration options

- `#define MFS_CFG_MAX_RECORDS 32`
Maximum number of indexed records in the managed storage.
- `#define MFS_CFG_MAX_REPAIR_ATTEMPTS 3`
Maximum number of repair attempts on partition mount.
- `#define MFS_CFG_WRITE_VERIFY TRUE`
Verify written data.
- `#define MFS_CFG_STRONG_CHECKING TRUE`
Enables a stronger and slower check procedure on mount.
- `#define MFS_CFG_BUFFER_SIZE 32`
Size of the buffer used for data copying.

Error codes handling macros

- `#define MFS_IS_ERROR(err) ((err) < MFS_NO_ERROR)`
- `#define MFS_IS_WARNING(err) ((err) > MFS_NO_ERROR)`

Typedefs

- `typedef uint32_t mfs_id_t`
Type of a record identifier.

Data Structures

- union [mfs_bank_header_t](#)
Type of a bank header.
- union [mfs_data_header_t](#)
Type of a data block header.
- struct [MFSCConfig](#)
Type of a MFS configuration structure.
- struct [MFSDriver](#)
Type of an MFS instance.

Functions

- static [mfs_error_t](#) [mfs_flash_read](#) ([MFSDriver](#) *mfsp, [flash_offset_t](#) offset, [size_t](#) n, [uint8_t](#) *rp)
Flash read.
- static [mfs_error_t](#) [mfs_flash_write](#) ([MFSDriver](#) *mfsp, [flash_offset_t](#) offset, [size_t](#) n, const [uint8_t](#) *wp)
Flash write.
- static [mfs_error_t](#) [mfs_flash_copy](#) ([MFSDriver](#) *mfsp, [flash_offset_t](#) doffset, [flash_offset_t](#) soffset, [uint32_t](#) n)
Flash copy.
- static [mfs_error_t](#) [mfs_record_check](#) ([MFSDriver](#) *mfsp, [mfs_data_header_t](#) *dhdrp, [flash_offset_t](#) offset, [flash_offset_t](#) limit, [mfs_record_state_t](#) *sts)
Verifies integrity of a record.
- static [mfs_error_t](#) [mfs_bank_erase](#) ([MFSDriver](#) *mfsp, [mfs_bank_t](#) bank)
Erases and verifies all sectors belonging to a bank.
- static [mfs_error_t](#) [mfs_bank_verify_erase](#) ([MFSDriver](#) *mfsp, [mfs_bank_t](#) bank)
Erases and verifies all sectors belonging to a bank.
- static [mfs_error_t](#) [mfs_bank_write_header](#) ([MFSDriver](#) *mfsp, [mfs_bank_t](#) bank, [uint32_t](#) cnt)
Writes the validation header in a bank.
- static [mfs_error_t](#) [mfs_bank_scan_records](#) ([MFSDriver](#) *mfsp, [mfs_bank_t](#) bank, [mfs_bank_state_t](#) *statep)
Scans blocks searching for records.
- static [mfs_error_t](#) [mfs_bank_get_state](#) ([MFSDriver](#) *mfsp, [mfs_bank_t](#) bank, [mfs_bank_state_t](#) *statep, [uint32_t](#) *cntp)
Determines the state of a bank.
- static [mfs_error_t](#) [mfs_bank_mount](#) ([MFSDriver](#) *mfsp, [mfs_bank_t](#) bank, [mfs_bank_state_t](#) *statep)
Selects a bank as current.
- static [mfs_error_t](#) [mfs_garbage_collect](#) ([MFSDriver](#) *mfsp)
Enforces a garbage collection.
- static [mfs_error_t](#) [mfs_try_mount](#) ([MFSDriver](#) *mfsp)
Performs a flash partition mount attempt.
- [mfs_error_t](#) [mfs_mount](#) ([MFSDriver](#) *mfsp)
Configures and activates a MFS driver.
- void [mfsObjectInit](#) ([MFSDriver](#) *mfsp)
Initializes an instance.
- [mfs_error_t](#) [mfsStart](#) ([MFSDriver](#) *mfsp, const [MFSCConfig](#) *config)
Configures and activates a MFS driver.
- void [mfsStop](#) ([MFSDriver](#) *mfsp)
Deactivates a MFS driver.
- [mfs_error_t](#) [mfsErase](#) ([MFSDriver](#) *mfsp)
Destroys the state of the managed storage by erasing the flash.
- [mfs_error_t](#) [mfsReadRecord](#) ([MFSDriver](#) *mfsp, [mfs_id_t](#) id, [size_t](#) *np, [uint8_t](#) *buffer)
Retrieves and reads a data record.

- `mfs_error_t mfsWriteRecord (MFSDriver *mfsp, mfs_id_t id, size_t n, const uint8_t *buffer)`
Creates or updates a data record.
- `mfs_error_t mfsEraseRecord (MFSDriver *mfsp, mfs_id_t id)`
Erases a data record.
- `mfs_error_t mfsPerformGarbageCollection (MFSDriver *mfsp)`
Enforces a garbage collection operation.

Enumerations

7.35.2 Macro Definition Documentation

7.35.2.1 `#define RET_ON_ERROR(err)`

Value:

```
do {
    mfs_error_t e = (err);
    if (e != MFS_NO_ERROR) {
        return e;
    }
} while (false)
```

Error check helper.

7.35.2.2 `#define MFS_CFG_MAX_RECORDS 32`

Maximum number of indexed records in the managed storage.

Note

Record indexes go from 0 to `MFS_CFG_MAX_RECORDS - 1`.

7.35.2.3 `#define MFS_CFG_MAX_REPAIR_ATTEMPTS 3`

Maximum number of repair attempts on partition mount.

7.35.2.4 `#define MFS_CFG_WRITE_VERIFY TRUE`

Verify written data.

7.35.2.5 `#define MFS_CFG_STRONG_CHECKING TRUE`

Enables a stronger and slower check procedure on mount.

Strong checking requires reading of the whole written data and this can be slow, normal checking only checks integrity of metadata, data errors would be detected on read.

7.35.2.6 `#define MFS_CFG_BUFFER_SIZE 32`

Size of the buffer used for data copying.

Note

The buffer size must be a power of two and not smaller than 16 bytes.

Larger buffers improve performance, buffers with size multiple of the flash program page size work better.

7.35.3 Typedef Documentation

7.35.3.1 typedef uint32_t mfs_id_t

Type of a record identifier.

7.35.4 Enumeration Type Documentation

7.35.4.1 enum mfs_bank_t

Type of a flash bank.

7.35.4.2 enum mfs_state_t

Type of driver state machine states.

7.35.4.3 enum mfs_error_t

Type of an MFS error code.

Note

Errors are negative integers, informative warnings are positive integers.

7.35.4.4 enum mfs_bank_state_t

Type of a bank state assessment.

7.35.4.5 enum mfs_record_state_t

Type of a record state assessment.

7.35.5 Function Documentation

7.35.5.1 static mfs_error_t mfs_flash_read (MFSDriver * *mfs*, flash_offset_t *offset*, size_t *n*, uint8_t * *rp*) [static]

Flash read.

Parameters

in	<i>mfs</i>	pointer to the MFSDriver object
in	<i>offset</i>	flash offset
in	<i>n</i>	number of bytes to be read
out	<i>rp</i>	pointer to the data buffer

Returns

The operation status.

Function Class:

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

7.35.5.2 `static mfs_error_t mfs_flash_write (MFSDriver * mfsp, flash_offset_t offset, size_t n, const uint8_t * wp)`
`[static]`

Flash write.

Note

If the option `MFS_CFG_WRITE_VERIFY` is enabled then the flash is also read back for verification.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>offset</i>	flash offset
in	<i>n</i>	number of bytes to be written
in	<i>wp</i>	pointer to the data buffer

Returns

The operation status.

Function Class:

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

Here is the call graph for this function:



7.35.5.3 `static mfs_error_t mfs_flash_copy (MFSDriver * mfsp, flash_offset_t doffset, flash_offset_t soffset, uint32_t n)`
`[static]`

Flash copy.

Note

If the option `MFS_CFG_WRITE_VERIFY` is enabled then the flash is also read back for verification.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>doffset</i>	destination flash offset
in	<i>soffset</i>	source flash offset
in	<i>n</i>	number of bytes to be copied

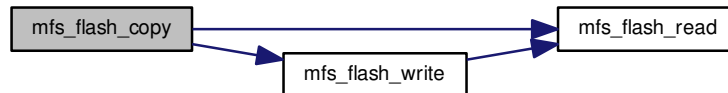
Returns

The operation status.

Function Class:

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

Here is the call graph for this function:



7.35.5.4 `static mfs_error_t mfs_record_check (MFSDriver * mfsp, mfs_data_header_t * dhdrp, flash_offset_t offset, flash_offset_t limit, mfs_record_state_t * sts) [static]`

Verifies integrity of a record.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>dhdrp</i>	pointer to the header to be checked
in	<i>offset</i>	flash offset of the header to be checked
in	<i>limit</i>	flash limit offset
out	<i>sts</i>	assessed record state

Returns

The operation status.

Function Class:

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

7.35.5.5 `static mfs_error_t mfs_bank_erase (MFSDriver * mfsp, mfs_bank_t bank) [static]`

Erases and verifies all sectors belonging to a bank.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>bank</i>	bank to be erased

Returns

The operation status.

Function Class:

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

Here is the call graph for this function:



7.35.5.6 `static mfs_error_t mfs_bank_verify_erase (MFSDriver * mfsp, mfs_bank_t bank) [static]`

Erases and verifies all sectors belonging to a bank.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>bank</i>	bank to be verified

Returns

The operation status.

Function Class:

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

7.35.5.7 `static mfs_error_t mfs_bank_write_header (MFSDriver * mfsp, mfs_bank_t bank, uint32_t cnt) [static]`

Writes the validation header in a bank.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>bank</i>	bank to be validated
in	<i>cnt</i>	value for the flash usage counter

Returns

The operation status.

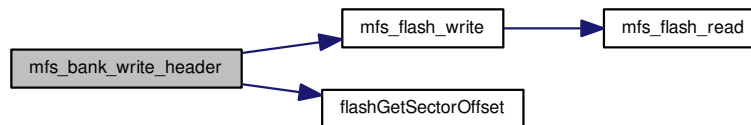
Return values

<code>MFS_NO_ERROR</code>	if the operation has been successfully completed.
---------------------------	---

Function Class:

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

Here is the call graph for this function:



7.35.5.8 `static mfs_error_t mfs_bank_scan_records (MFSDriver * mfsp, mfs_bank_t bank, mfs_bank_state_t * statep)` [static]

Scans blocks searching for records.

Note

The block integrity is strongly checked.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>bank</i>	the bank identifier
out	<i>statep</i>	bank state, it can be: <ul style="list-style-type: none"> • MFS_BANK_PARTIAL • MFS_BANK_OK

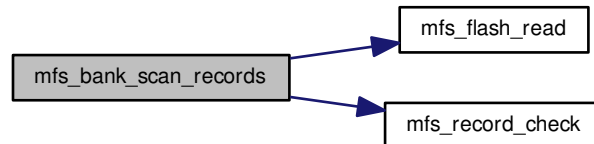
Returns

The operation status.

Function Class:

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

Here is the call graph for this function:



7.35.5.9 `static mfs_error_t mfs_bank_get_state (MFSDriver * mfsp, mfs_bank_t bank, mfs_bank_state_t * statep, uint32_t * cntp) [static]`

Determines the state of a bank.

Note

This function does not test the bank integrity by scanning the data area, it just checks the header.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>bank</i>	bank to be checked
out	<i>statep</i>	bank state, it can be: <ul style="list-style-type: none"> • MFS_BANK_ERASED • MFS_BANK_GARBAGE • MFS_BANK_OK
out	<i>cntp</i>	bank counter

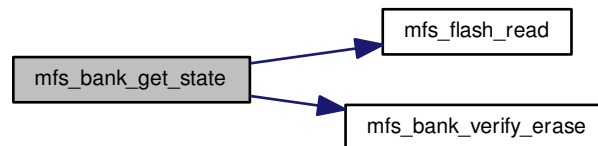
Returns

The operation status.

Function Class:

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

Here is the call graph for this function:



7.35.5.10 `static mfs_error_t mfs_bank_mount (MFSDriver * mfsp, mfs_bank_t bank, mfs_bank_state_t * statep)`
[static]

Selects a bank as current.

Note

The bank header is assumed to be valid.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>bank</i>	bank to be scanned
out	<i>statep</i>	bank state, it can be: <ul style="list-style-type: none">• MFS_BANK_ERASED• MFS_BANK_GARBAGE• MFS_BANK_PARTIAL• MFS_BANK_OK

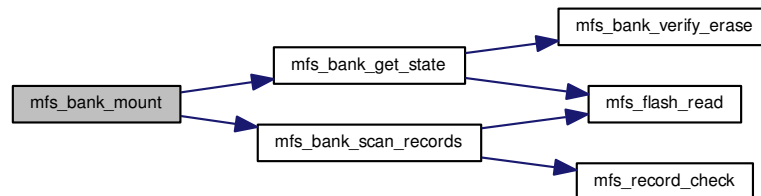
Returns

The operation status.

Function Class:

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

Here is the call graph for this function:



7.35.5.11 `static mfs_error_t mfs_garbage_collect (MFSDriver * mfsp) [static]`

Enforces a garbage collection.

Storage data is compacted into a single bank.

Parameters

out	<i>mfsp</i>	pointer to the MFSDriver object
-----	-------------	---

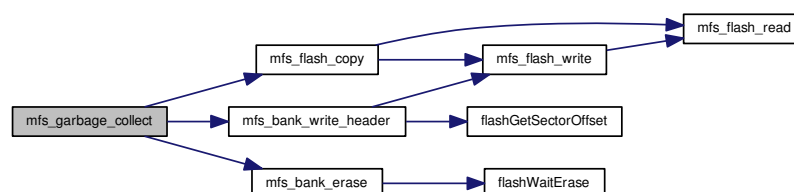
Returns

The operation status.

Function Class:

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

Here is the call graph for this function:



7.35.5.12 static mfs_error_t mfs_try_mount (MFSDriver * mfsp) [static]

Performs a flash partition mount attempt.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
----	-------------	---

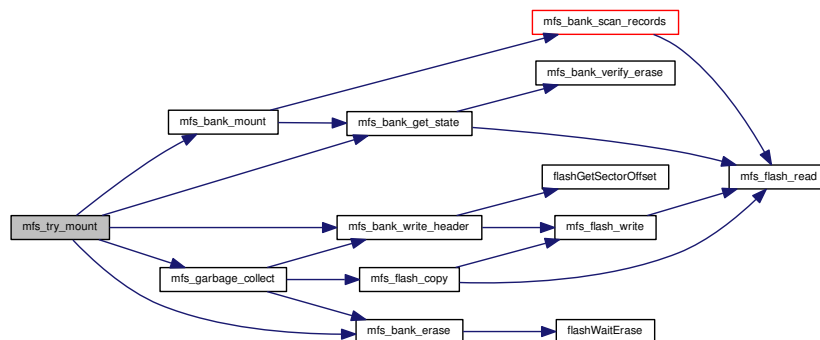
Returns

The operation status.

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:



7.35.5.13 mfs_error_t mfs_mount (MFSDriver * mfsp)

Configures and activates a MFS driver.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
----	-------------	---

Returns

The operation status.

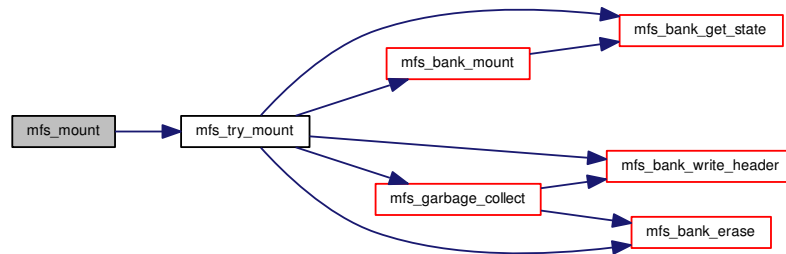
Return values

<i>MFS_NO_ERROR</i>	if the operation has been successfully completed.
<i>MFS_WARN_GC</i>	if the operation triggered a garbage collection.
<i>MFS_ERR_FLASH_FAILURE</i>	if the flash memory is unusable because HW failures. Makes the driver enter the <i>MFS_ERROR</i> state.
<i>MFS_ERR_INTERNAL</i>	if an internal logic failure is detected.

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:



7.35.5.14 void mfsObjectInit (MFSDriver * mfsp)

Initializes an instance.

Parameters

out	mfsp	pointer to the MFSDriver object
-----	------	---

Function Class:

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

7.35.5.15 mfs_error_t mfsStart (MFSDriver * mfsp, const MFSCfg * config)

Configures and activates a MFS driver.

Parameters

in	mfsp	pointer to the MFSDriver object
in	config	pointer to the configuration

Returns

The operation status.

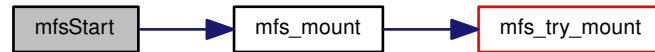
Return values

<i>MFS_NO_ERROR</i>	if the operation has been successfully completed.
<i>MFS_WARN_GC</i>	if the operation triggered a garbage collection.
<i>MFS_ERR_FLASH_FAILURE</i>	if the flash memory is unusable because HW failures. Makes the driver enter the <i>MFS_ERROR</i> state.
<i>MFS_ERR_INTERNAL</i>	if an internal logic failure is detected.

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:



7.35.5.16 void mfsStop (MFSDriver * mfsp)

Deactivates a MFS driver.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
----	-------------	---

Function Class:

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

7.35.5.17 mfs_error_t mfsErase (MFSDriver * mfsp)

Destroys the state of the managed storage by erasing the flash.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
----	-------------	---

Returns

The operation status.

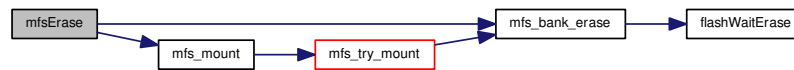
Return values

<i>MFS_ERR_INV_STATE</i>	if the driver is in not in MSG_READY state.
<i>MFS_NO_ERROR</i>	if the operation has been successfully completed.
<i>MFS_ERR_FLASH_FAILURE</i>	if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.
<i>MFS_ERR_INTERNAL</i>	if an internal logic failure is detected.

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:



7.35.5.18 mfs_error_t mfsReadRecord (MFSDriver * mfsp, mfs_id_t id, size_t * np, uint8_t * buffer)

Retrieves and reads a data record.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>id</i>	record numeric identifier, the valid range is between 1 and MFS_CFG_MAX_RECORDS
in, out	<i>np</i>	on input is the maximum buffer size, on return it is the size of the data copied into the buffer
out	<i>buffer</i>	pointer to a buffer for record data

Returns

The operation status.

Return values

<i>MFS_NO_ERROR</i>	if the operation has been successfully completed.
<i>MFS_ERR_INV_STATE</i>	if the driver is in not in MSG_READY state.
<i>MFS_ERR_INV_SIZE</i>	if the passed buffer is not large enough to contain the record data.
<i>MFS_ERR_NOT_FOUND</i>	if the specified id does not exists.
<i>MFS_ERR_FLASH_FAILURE</i>	if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.
<i>MFS_ERR_INTERNAL</i>	if an internal logic failure is detected.

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:



7.35.5.19 mfs_error_t mfsWriteRecord (MFSDriver * mfsp, mfs_id_t id, size_t n, const uint8_t * buffer)

Creates or updates a data record.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>id</i>	record numeric identifier, the valid range is between 1 and MFS_CFG_MAX_RECORDS
in	<i>n</i>	size of data to be written, it cannot be zero
in	<i>buffer</i>	pointer to a buffer for record data

Returns

The operation status.

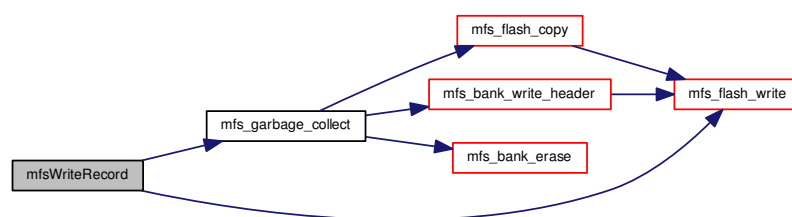
Return values

<i>MFS_NO_ERROR</i>	if the operation has been successfully completed.
<i>MFS_WARN_GC</i>	if the operation triggered a garbage collection.
<i>MFS_ERR_INV_STATE</i>	if the driver is in not in MSG_READY state.
<i>MFS_ERR_OUT_OF_MEM</i>	if there is not enough flash space for the operation.
<i>MFS_ERR_FLASH_FAILURE</i>	if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.
<i>MFS_ERR_INTERNAL</i>	if an internal logic failure is detected.

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:



7.35.5.20 mfs_error_t mfsEraseRecord (MFSDriver * mfsp, mfs_id_t id)

Erases a data record.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
in	<i>id</i>	record numeric identifier, the valid range is between 1 and MFS_CFG_MAX_RECORDS

Returns

The operation status.

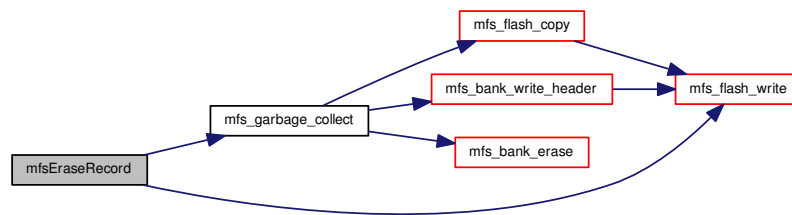
Return values

<i>MFS_NO_ERROR</i>	if the operation has been successfully completed.
<i>MFS_WARN_GC</i>	if the operation triggered a garbage collection.
<i>MFS_ERR_INV_STATE</i>	if the driver is in not in MSG_READY state.
<i>MFS_ERR_FLASH_FAILURE</i>	if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.
<i>MFS_ERR_INTERNAL</i>	if an internal logic failure is detected.

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:

**7.35.5.21 mfs_error_t mfsPerformGarbageCollection (MFSDriver * mfsp)**

Enforces a garbage collection operation.

Garbage collection involves: integrity check, optionally repairs, obsolete data removal, data compaction and a flash bank swap.

Parameters

in	<i>mfsp</i>	pointer to the MFSDriver object
----	-------------	---------------------------------

Returns

The operation status.

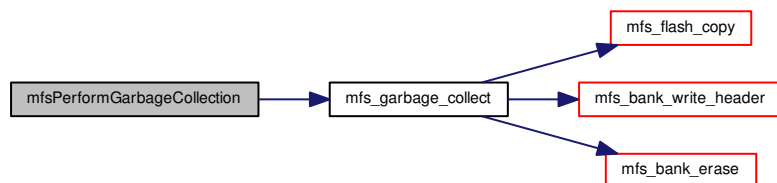
Return values

<i>MFS_NO_ERROR</i>	if the operation has been successfully completed.
<i>MFS_ERR_INV_STATE</i>	if the driver is in not in MSG_READY state.
<i>MFS_ERR_FLASH_FAILURE</i>	if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.
<i>MFS_ERR_INTERNAL</i>	if an internal logic failure is detected.

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 MII/RMII Header

MIIRMII Support Header.

7.36.1 Detailed Description

MIIRMII 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` 0x0a
- #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_LBRRERROR` 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_KSZ8081_ID` 0x00221560
- #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

7.36.2 Macro Definition Documentation**7.36.2.1 #define MII_BMCR 0x00**

Basic mode control register.

7.36.2.2 #define MII_BMSR 0x01

Basic mode status register.

7.36.2.3 #define MII_PHYSID1 0x02

PHYS ID 1.

7.36.2.4 #define MII_PHYSID2 0x03

PHYS ID 2.

7.36.2.5 #define MII_ADVERTISE 0x04

Advertisement control reg.

7.36.2.6 #define MII_LPA 0x05

Link partner ability reg.

7.36.2.7 #define MII_EXPANSION 0x06

Expansion register.

7.36.2.8 #define MII_ANNPTR 0x07

1000BASE-T control.

7.36.2.9 #define MII_CTRL1000 0x09

1000BASE-T control.

7.36.2.10 #define MII_STAT1000 0x0a

1000BASE-T status.

7.36.2.11 #define MII_ESTATUS 0x0f

Extended Status.

7.36.2.12 #define MII_PHYSTS 0x10

PHY Status register.

7.36.2.13 #define MII_MICR 0x11

MII Interrupt ctrl register.

7.36.2.14 #define MII_DCOUNTER 0x12

Disconnect counter.

7.36.2.15 #define MII_FCSCOUNTER 0x13

False carrier counter.

7.36.2.16 #define MII_NWAYTEST 0x14

N-way auto-neg test reg.

7.36.2.17 #define MII_RERRCOUNTER 0x15

Receive error counter.

7.36.2.18 #define MII_SREVISION 0x16

Silicon revision.

7.36.2.19 #define MII_RESV1 0x17

Reserved.

7.36.2.20 #define MII_LBRERROR 0x18

Lpback, rx, bypass error.

7.36.2.21 #define MII_PHYADDR 0x19

PHY address.

7.36.2.22 #define MII_RESV2 0x1a

Reserved.

7.36.2.23 #define MII_TPISTATUS 0x1b

TPI status for 10Mbps.

7.36.2.24 #define MII_NCONFIG 0x1c

Network interface config.

7.36.2.25 #define BMCR_RESV 0x007f

Unused.

7.36.2.26 #define BMCR_CTST 0x0080

Collision test.

7.36.2.27 #define BMCR_FULLDPLX 0x0100

Full duplex.

7.36.2.28 #define BMCR_ANRESTART 0x0200

Auto negotiation restart.

7.36.2.29 #define BMCR_ISOLATE 0x0400

Disconnect DP83840 from MII.

7.36.2.30 `#define BMCR_PDOWN 0x0800`

Powerdown.

7.36.2.31 `#define BMCR_ANENABLE 0x1000`

Enable auto negotiation.

7.36.2.32 `#define BMCR_SPEED100 0x2000`

Select 100Mbps.

7.36.2.33 `#define BMCR_LOOPBACK 0x4000`

TXD loopback bit.

7.36.2.34 `#define BMCR_RESET 0x8000`

Reset.

7.36.2.35 `#define BMSR_ERCAP 0x0001`

Ext-reg capability.

7.36.2.36 `#define BMSR_JCD 0x0002`

Jabber detected.

7.36.2.37 `#define BMSR_LSTATUS 0x0004`

Link status.

7.36.2.38 `#define BMSR_ANEGCAPABLE 0x0008`

Able to do auto-negotiation.

7.36.2.39 `#define BMSR_RFAULT 0x0010`

Remote fault detected.

7.36.2.40 `#define BMSR_ANEGCOMPLETE 0x0020`

Auto-negotiation complete.

7.36.2.41 `#define BMSR_MFPRESUPPCAP 0x0040`

Able to suppress preamble.

7.36.2.42 `#define BMSR_RESV 0x0780`

Unused.

7.36.2.43 `#define BMSR_10HALF 0x0800`

Can do 10mbps, half-duplex.

7.36.2.44 `#define BMSR_10FULL 0x1000`

Can do 10mbps, full-duplex.

7.36.2.45 `#define BMSR_100HALF 0x2000`

Can do 100mbps, half-duplex.

7.36.2.46 `#define BMSR_100FULL 0x4000`

Can do 100mbps, full-duplex.

7.36.2.47 `#define BMSR_100BASE4 0x8000`

Can do 100mbps, 4k packets.

7.36.2.48 `#define ADVERTISE_SLCT 0x001f`

Selector bits.

7.36.2.49 `#define ADVERTISE_CSMA 0x0001`

Only selector supported.

7.36.2.50 `#define ADVERTISE_10HALF 0x0020`

Try for 10mbps half-duplex.

7.36.2.51 `#define ADVERTISE_10FULL 0x0040`

Try for 10mbps full-duplex.

7.36.2.52 `#define ADVERTISE_100HALF 0x0080`

Try for 100mbps half-duplex.

7.36.2.53 `#define ADVERTISE_100FULL 0x0100`

Try for 100mbps full-duplex.

7.36.2.54 `#define ADVERTISE_100BASE4 0x0200`

Try for 100mbps 4k packets.

7.36.2.55 `#define ADVERTISE_PAUSE_CAP 0x0400`

Try for pause.

7.36.2.56 `#define ADVERTISE_PAUSE_ASYM 0x0800`

Try for asymmetric pause.

7.36.2.57 `#define ADVERTISE_RESV 0x1000`

Unused.

7.36.2.58 `#define ADVERTISE_RFAULT 0x2000`

Say we can detect faults.

7.36.2.59 `#define ADVERTISE_LPACK 0x4000`

Ack link partners response.

7.36.2.60 `#define ADVERTISE_NPAGE 0x8000`

Next page bit.

7.36.2.61 `#define LPA_SLCT 0x001f`

Same as advertise selector.

7.36.2.62 `#define LPA_10HALF 0x0020`

Can do 10mbps half-duplex.

7.36.2.63 `#define LPA_10FULL 0x0040`

Can do 10mbps full-duplex.

7.36.2.64 `#define LPA_100HALF 0x0080`

Can do 100mbps half-duplex.

7.36.2.65 `#define LPA_100FULL 0x0100`

Can do 100mbps full-duplex.

7.36.2.66 `#define LPA_100BASE4 0x0200`

Can do 100mbps 4k packets.

7.36.2.67 `#define LPA_PAUSE_CAP 0x0400`

Can pause.

7.36.2.68 `#define LPA_PAUSE_ASYM 0x0800`

Can pause asymmetrically.

7.36.2.69 `#define LPA_RESV 0x1000`

Unused.

7.36.2.70 `#define LPA_RFAULT 0x2000`

Link partner faulted.

7.36.2.71 `#define LPA_LPACK 0x4000`

Link partner acked us.

7.36.2.72 `#define LPA_NPAGE 0x8000`

Next page bit.

7.36.2.73 `#define EXPANSION_NWAY 0x0001`

Can do N-way auto-nego.

7.36.2.74 `#define EXPANSION_LCWP 0x0002`

Got new RX page code word.

7.36.2.75 `#define EXPANSION_ENABLENPAGE 0x0004`

This enables npage words.

7.36.2.76 `#define EXPANSION_NPCAPABLE 0x0008`

Link partner supports npage.

7.36.2.77 `#define EXPANSION_MFAULTS 0x0010`

Multiple faults detected.

7.36.2.78 `#define EXPANSION_RESV 0xffe0`

Unused.

7.36.2.79 `#define NWAYTEST_RESV1 0x00ff`

Unused.

7.36.2.80 `#define NWAYTEST_LOOPBACK 0x0100`

Enable loopback for N-way.

7.36.2.81 `#define NWAYTEST_RESV2 0xfe00`

Unused.

7.37 MMC over SPI Driver

Generic MMC driver.

7.37.1 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`.

7.37.2 Driver State Machine

This driver implements a state machine internally, see the [Abstract I/O Block Device](#) module documentation for details.

7.37.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Macros

- `#define _mmc_driver_methods _mmc_sd_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 `MMConfig`
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 **recvr1** (MMCDriver *mmcp)
Receives a single byte response.
- static uint8_t **recvr3** (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 **mmclnit** (void)
MMC over SPI driver initialization.
- void **mmcObjectInit** (MMCDriver *mmcp)
Initializes an instance.
- void **mmcStart** (MMCDriver *mmcp, const MMCCfg *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 $x^7 + x^3 + 1$).

7.37.4 Macro Definition Documentation

7.37.4.1 #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.

7.37.4.2 #define _mmc_driver_methods _mmc_ssd_block_device_methods

[MMCDriver](#) specific methods.

7.37.4.3 #define mmclsCardInserted(mmcp) mmc_lld_is_card_inserted(mmcp)

Returns the card insertion status.

Note

This macro wraps a low level function named `sd_c_lld_is_card_inserted()`, this function must be provided by the application because it is not part of the SDC driver.

Parameters

in	<code>mmcp</code>	pointer to the MMCDriver object
----	-------------------	---

Returns

The card state.

Return values

<code>false</code>	card not inserted.
<code>true</code>	card inserted.

Function Class:

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

7.37.4.4 #define mmclsWriteProtected(mmcp) mmc_lld_is_write_protected(mmcp)

Returns the write protect status.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver 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.

7.37.5 Function Documentation**7.37.5.1** `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

7.37.5.2 `static void wait (MMCDriver* mmcp)` `[static]`

Waits an idle condition.

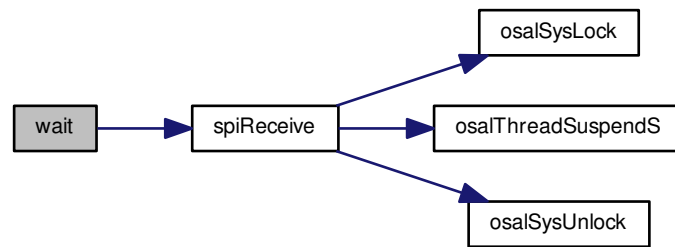
Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.37.5.3 `static void send_hdr (MMCDriver * mmcp, uint8_t cmd, uint32_t arg) [static]`

Sends a command header.

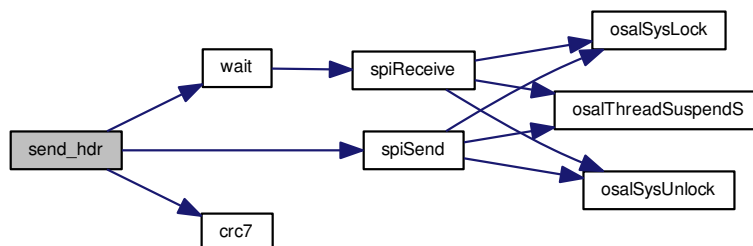
Parameters

in	<i>mmcp</i>	pointer to the MMCDriver 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:



7.37.5.4 `static uint8_t recvr1 (MMCDriver * mmcp) [static]`

Receives a single byte response.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The response as an `uint8_t` value.

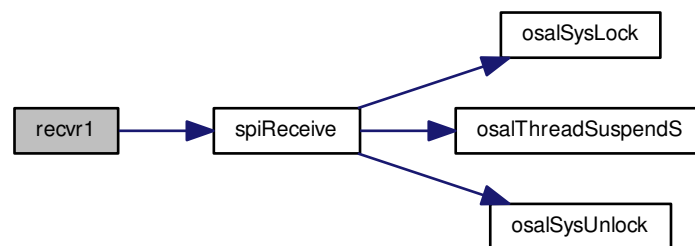
Return values

<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:



7.37.5.5 `static uint8_t recvr3 (MMCDriver * mmcp, uint8_t * buffer) [static]`

Receives a three byte response.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
out	<i>buffer</i>	pointer to four bytes wide buffer

Returns

First response byte as an `uint8_t` value.

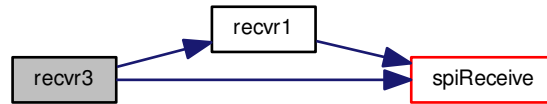
Return values

<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:



7.37.5.6 `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 MMCDriver object
in	<i>cmd</i>	the command id
in	<i>arg</i>	the command argument

Returns

The response as an `uint8_t` value.

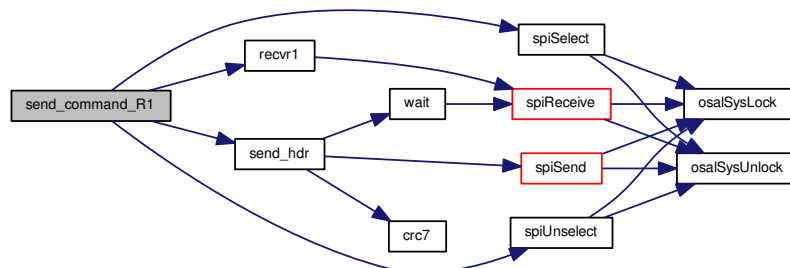
Return values

<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:



7.37.5.7 `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 MMCDriver object
in	<i>cmd</i>	the command id
in	<i>arg</i>	the command argument
out	<i>response</i>	pointer to four bytes wide <code>uint8_t</code> buffer

Returns

The first byte of the response (R1) as an `uint8_t` value.

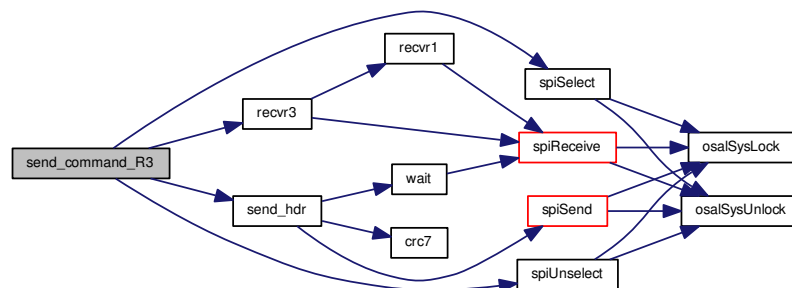
Return values

<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:



7.37.5.8 `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 MMCDriver object
out	<i>cmd</i>	command
out	<i>cxd</i>	pointer to the CSD/CID buffer

Returns

The operation status.

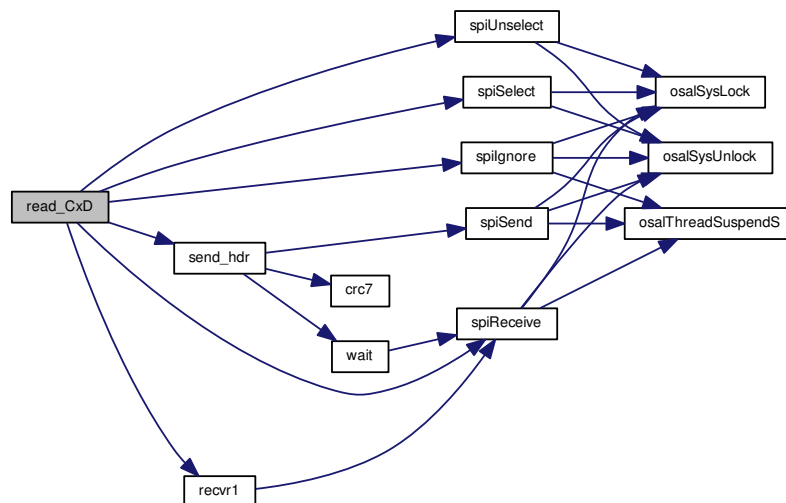
Return values

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	the operation failed.

Function Class:

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

Here is the call graph for this function:

**7.37.5.9 static void sync (MMCDriver * mmcp) [static]**

Waits that the card reaches an idle state.

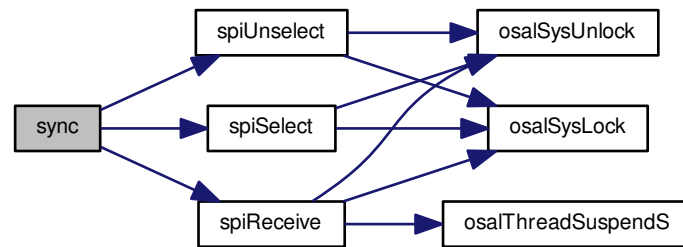
Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.37.5.10 void mmclnit (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.

7.37.5.11 void mmcObjectInit (MMCDriver * mmcp)

Initializes an instance.

Parameters

out	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
-----	-------------	--

Function Class:

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

7.37.5.12 void mmcStart (MMCDriver * mmcp, const MMCCConfig * config)

Configures and activates the MMC peripheral.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
in	<i>config</i>	pointer to the <code>MMCCConfig</code> object.

Function Class:

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

7.37.5.13 void mmcStop (MMCDriver * mmcp)

Disables the MMC peripheral.

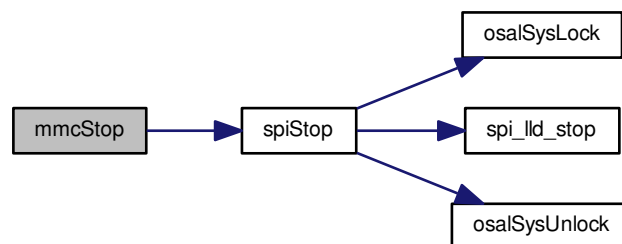
Parameters

in	<i>mmcp</i>	pointer to the MMCDriver 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:

**7.37.5.14 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

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The operation status.

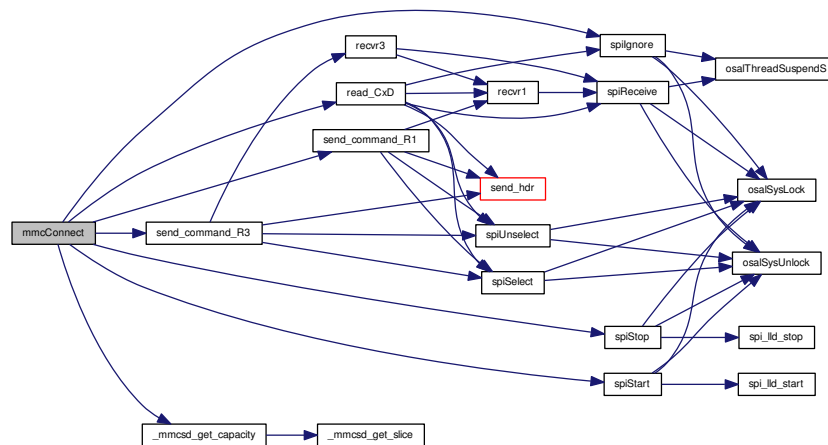
Return values

<i>HAL_SUCCESS</i>	the operation succeeded and the driver is now in the MMC_READY 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:



7.37.5.15 bool mmcDisconnect (MMCDriver * mmcp)

Brings the driver in a state safe for card removal.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The operation status.

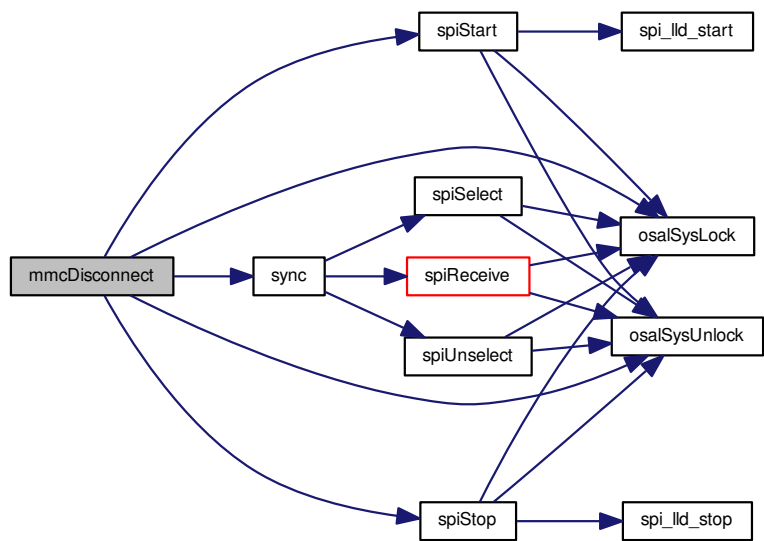
Return values

<i>HAL_SUCCESS</i>	the operation succeeded and the driver is now in the MMC_INSERTED 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:



7.37.5.16 `bool mmcStartSequentialRead (MMCDriver * mmcp, uint32_t startblk)`

Starts a sequential read.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
in	<i>startblk</i>	first block to read

Returns

The operation status.

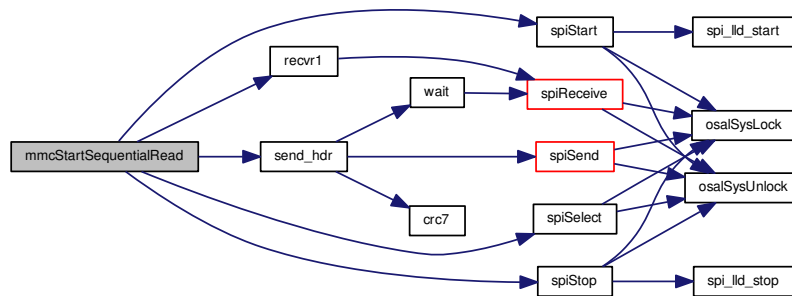
Return values

<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:



7.37.5.17 `bool mmcSequentialRead (MMCDriver * mmcp, uint8_t * buffer)`

Reads a block within a sequential read operation.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
out	<i>buffer</i>	pointer to the read buffer

Returns

The operation status.

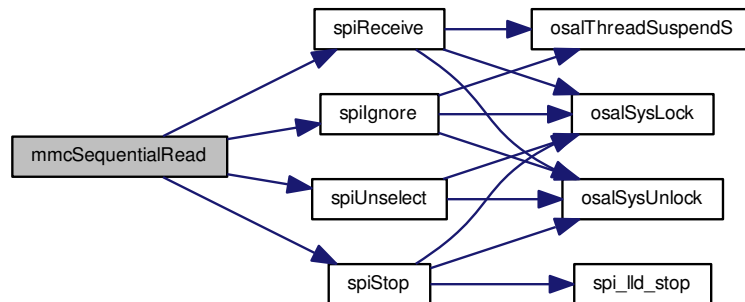
Return values

<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:



7.37.5.18 bool mmcStopSequentialRead (MMCDriver * mmcp)

Stops a sequential read gracefully.

Parameters

in	mmcp	pointer to the MMCDriver object
----	------	---

Returns

The operation status.

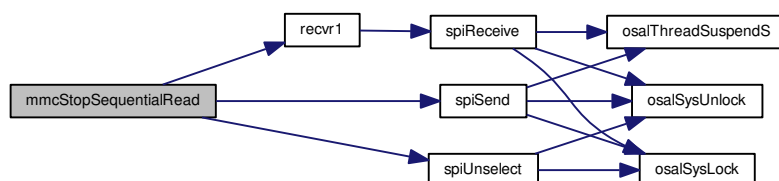
Return values

<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:



7.37.5.19 bool mmcStartSequentialWrite (MMCDriver * mmcp, uint32_t startblk)

Starts a sequential write.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
in	<i>startblk</i>	first block to write

Returns

The operation status.

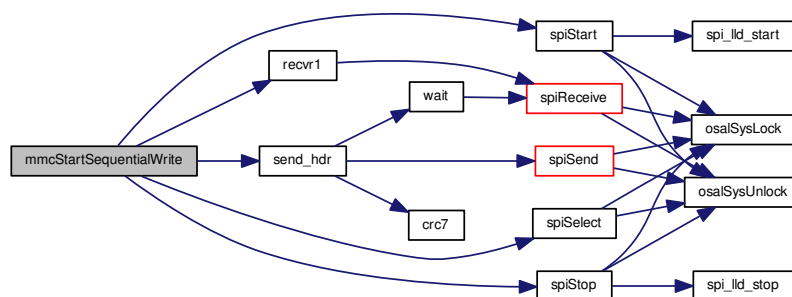
Return values

<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:



7.37.5.20 bool mmcSequentialWrite (MMCDriver * mmcp, const uint8_t * buffer)

Writes a block within a sequential write operation.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
out	<i>buffer</i>	pointer to the write buffer

Returns

The operation status.

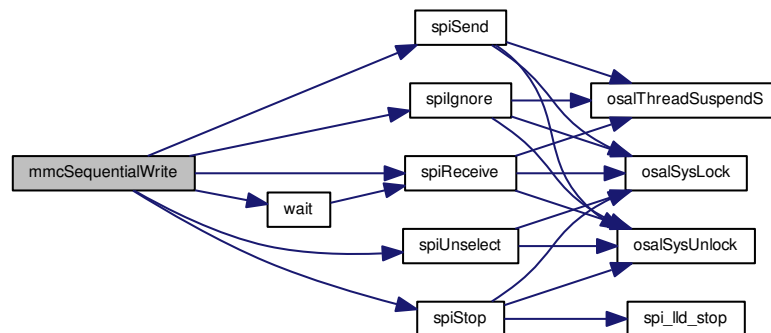
Return values

<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:



7.37.5.21 bool mmcStopSequentialWrite (MMCDriver * mmcp)

Stops a sequential write gracefully.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The operation status.

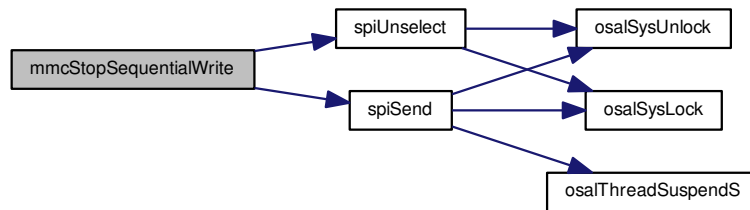
Return values

<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:



7.37.5.22 bool mmcSync (MMCDriver * mmcp)

Waits for card idle condition.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The operation status.

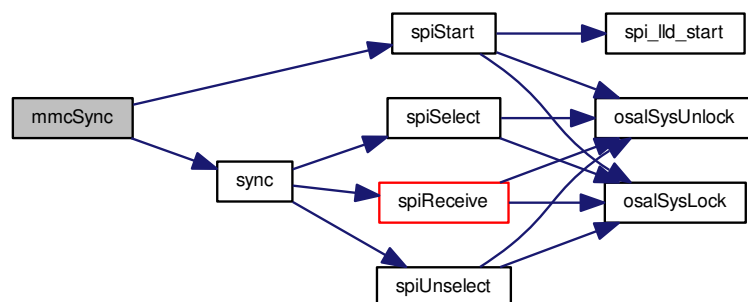
Return values

<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:



7.37.5.23 `bool mmcGetInfo (MMCDriver * mmcp, BlockDeviceInfo * bdip)`

Returns the media info.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
out	<i>bdip</i>	pointer to a BlockDeviceInfo structure

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.

7.37.5.24 `bool mmcErase (MMCDriver * mmcp, uint32_t startblk, uint32_t endblk)`

Erases blocks.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
in	<i>startblk</i>	starting block number
in	<i>endblk</i>	ending block number

Returns

The operation status.

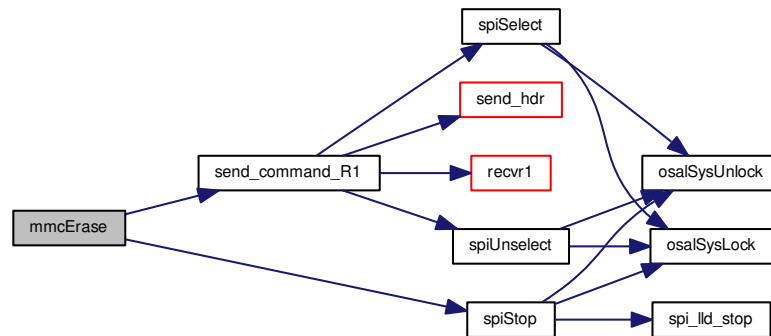
Return values

<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:



7.37.6 Variable Documentation

7.37.6.1 const struct MMCDriverVMT mmc_vmt [static]

Initial value:

```

= {
    (size_t)0,
    (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.

7.37.6.2 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, 0x0b, 0x02, 0x3d, 0x34, 0x2f, 0x26,
    0x51, 0x58, 0x43, 0x4a, 0x75, 0x7c, 0x67, 0x6e, 0x32, 0x3b, 0x20, 0x29,
    0x16, 0x1f, 0x04, 0x0d, 0x7a, 0x73, 0x68, 0x61, 0x5e, 0x57, 0x4c, 0x45,
    0x2b, 0x22, 0x39, 0x30, 0x0f, 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, 0x0a,
    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, 0x0e, 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 $x^7 + x^3 + 1$).

7.38 MMC/SD Block Device

7.38.1 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 0xFDFFE008U`
Mask of error bits in R1 responses.
- `#define MMCSD_CMD8_PATTERN 0x000001AAU`
Fixed pattern for CMD8.
- `#define _mmcscd_block_device_methods _base_block_device_methods`
MMCSDBlockDevice specific methods.
- `#define _mmcscd_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 `MMCSA_CID_SDC_CRC_SLICE` 7U,1U
Slice position of values in CID register.
- #define `MMCSA_CID_SDC_MDT_M_SLICE` 11U,8U
- #define `MMCSA_CID_SDC_MDT_Y_SLICE` 19U,12U
- #define `MMCSA_CID_SDC_PSN_SLICE` 55U,24U
- #define `MMCSA_CID_SDC_PRV_M_SLICE` 59U,56U
- #define `MMCSA_CID_SDC_PRV_N_SLICE` 63U,60U
- #define `MMCSA_CID_SDC_PNM0_SLICE` 71U,64U
- #define `MMCSA_CID_SDC_PNM1_SLICE` 79U,72U
- #define `MMCSA_CID_SDC_PNM2_SLICE` 87U,80U
- #define `MMCSA_CID_SDC_PNM3_SLICE` 95U,88U
- #define `MMCSA_CID_SDC_PNM4_SLICE` 103U,96U
- #define `MMCSA_CID_SDC_OID_SLICE` 119U,104U
- #define `MMCSA_CID_SDC_MID_SLICE` 127U,120U
- #define `MMCSA_CID_MMC_CRC_SLICE` 7U,1U
- #define `MMCSA_CID_MMC_MDT_Y_SLICE` 11U,8U
- #define `MMCSA_CID_MMC_MDT_M_SLICE` 15U,12U
- #define `MMCSA_CID_MMC_PSN_SLICE` 47U,16U
- #define `MMCSA_CID_MMC_PRV_M_SLICE` 51U,48U
- #define `MMCSA_CID_MMC_PRV_N_SLICE` 55U,52U
- #define `MMCSA_CID_MMC_PNM0_SLICE` 63U,56U
- #define `MMCSA_CID_MMC_PNM1_SLICE` 71U,64U
- #define `MMCSA_CID_MMC_PNM2_SLICE` 79U,72U
- #define `MMCSA_CID_MMC_PNM3_SLICE` 87U,80U
- #define `MMCSA_CID_MMC_PNM4_SLICE` 95U,88U
- #define `MMCSA_CID_MMC_PNM5_SLICE` 103U,96U
- #define `MMCSA_CID_MMC_OID_SLICE` 119U,104U
- #define `MMCSA_CID_MMC_MID_SLICE` 127U,120U

R1 response utilities

- #define `MMCSA_R1_ERROR(r1)` (((r1) & `MMCSA_R1_ERROR_MASK`) != 0U)
Evaluates to `true` if the R1 response contains error flags.
- #define `MMCSA_R1_STS(r1)` (((r1) >> 9U) & 15U)
Returns the status field of an R1 response.
- #define `MMCSA_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 CID register from MMC.
- struct `unpacked_sdc_csd_20_t`
Unpacked CSD v1.0 register from SDC.
- struct `unpacked_mmc_csd_t`
Unpacked CSD v2.0 register from SDC.
- struct `unpacked_mmc_csd_t`
Unpacked CSD register from MMC.

Functions

- `uint32_t _mmcscd_get_slice` (const `uint32_t` *data, `uint32_t` end, `uint32_t` start)
Gets a bit field from a words array.
- `uint32_t _mmcscd_get_capacity` (const `uint32_t` *csd)
Extract card capacity from a CSD.
- `uint32_t _mmcscd_get_capacity_ext` (const `uint8_t` *ext_csd)
Extract MMC card capacity from EXT_CSD.
- void `_mmcscd_unpack_sdc_cid` (const `MMCSDBlockDevice` *sdcp, `unpacked_sdc_cid_t` *cidsdc)
Unpacks SDC CID array in structure.
- void `_mmcscd_unpack_mmc_cid` (const `MMCSDBlockDevice` *sdcp, `unpacked_mmc_cid_t` *cidmmc)
Unpacks MMC CID array in structure.
- void `_mmcscd_unpack_csd_mmc` (const `MMCSDBlockDevice` *sdcp, `unpacked_mmc_csd_t` *csdmmc)
Unpacks MMC CSD array in structure.
- void `_mmcscd_unpack_csd_v10` (const `MMCSDBlockDevice` *sdcp, `unpacked_sdc_csd_10_t` *csd10)
Unpacks SDC CSD v1.0 array in structure.
- void `_mmcscd_unpack_csd_v20` (const `MMCSDBlockDevice` *sdcp, `unpacked_sdc_csd_20_t` *csd20)
Unpacks SDC CSD v2.0 array in structure.

7.38.2 Macro Definition Documentation

7.38.2.1 #define MMCSD_BLOCK_SIZE 512U

Fixed block size for MMC/SD block devices.

7.38.2.2 #define MMCSD_R1_ERROR_MASK 0xFDFFE008U

Mask of error bits in R1 responses.

7.38.2.3 #define MMCSD_CMD8_PATTERN 0x000001AAU

Fixed pattern for CMD8.

7.38.2.4 #define MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE 127U,126U

Slice position of values in CSD register.

7.38.2.5 #define MMCSD_CID_SDC_CRC_SLICE 7U,1U

Slice position of values in CID register.

7.38.2.6 `#define _mmcscd_block_device_methods _base_block_device_methods`

`MMCSDBlockDevice` specific methods.

7.38.2.7 `#define _mmcscd_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.

7.38.2.8 `#define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)`

Evaluates to `true` if the R1 response contains error flags.

Parameters

<code>in</code>	<code>r1</code>	the r1 response
-----------------	-----------------	-----------------

7.38.2.9 `#define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)`

Returns the status field of an R1 response.

Parameters

<code>in</code>	<code>r1</code>	the r1 response
-----------------	-----------------	-----------------

7.38.2.10 `#define MMCSD_R1_IS_CARD_LOCKED(r1) (((r1) >> 21U) & 1U) != 0U)`

Evaluates to `true` if the R1 response indicates a locked card.

Parameters

<code>in</code>	<code>r1</code>	the r1 response
-----------------	-----------------	-----------------

7.38.2.11 `#define mmcscdGetCardCapacity(ip) ((ip)->capacity)`

Returns the card capacity in blocks.

Parameters

in	<i>ip</i>	pointer to a MMCSDBlockDevice 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.

7.38.3 Function Documentation**7.38.3.1 uint32_t _mmcscd_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.

7.38.3.2 uint32_t _mmcscd_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:



7.38.3.3 `uint32_t _mmcscd_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.

7.38.3.4 `void _mmcscd_unpack_sdc_cid (const MMCSDBlockDevice * sdcp, unpacked_sdc_cid_t * cidsc)`

Unpacks SDC CID array in structure.

Parameters

in	<i>sdcp</i>	pointer to the MMCSDBlockDevice object
out	<i>cidsc</i>	pointer to the unpacked_sdc_cid_t object

Function Class:

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

Here is the call graph for this function:



7.38.3.5 void `_mmcscd_unpack_mmc_cid` (const `MMCSCDBlockDevice` * *sdc*, `unpacked_mmc_cid_t` * *cidmmc*)

Unpacks MMC CID array in structure.

Parameters

in	<i>sdc</i>	pointer to the MMCSCDBlockDevice object
out	<i>cidmmc</i>	pointer to the unpacked_mmc_cid_t object

Function Class:

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

Here is the call graph for this function:



7.38.3.6 void `_mmcscd_unpack_csd_mmc` (const `MMCSCDBlockDevice` * *sdc*, `unpacked_mmc_csd_t` * *csdmmc*)

Unpacks MMC CSD array in structure.

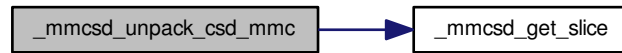
Parameters

in	<i>sdc</i>	pointer to the MMCSCDBlockDevice object
out	<i>csdmmc</i>	pointer to the unpacked_mmc_csd_t object

Function Class:

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

Here is the call graph for this function:



7.38.3.7 void _mmcscd_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 MMCSDBlockDevice object
out	<i>csd10</i>	pointer to the unpacked_sdc_csd_10_t object

Function Class:

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

Here is the call graph for this function:



7.38.3.8 void _mmcscd_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 MMCSDBlockDevice object
out	<i>csd20</i>	pointer to the unpacked_sdc_csd_20_t object

Function Class:

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

Here is the call graph for this function:



7.39 PAL Driver

I/O Ports Abstraction Layer.

7.39.1 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](#).

7.39.2 Implementation Rules

In implementing a PAL Low Level Driver there are some rules/behaviors that should be respected.

7.39.2.1 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.

7.39.2.2 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.

7.39.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

7.39.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

7.39.2.5 Reading or writing on pins associated to other functionalities

The behavior is not specified.

Macros

- `#define PAL_PORT_BIT(n) ((iportmask_t)(1U << (n)))`
Port bit helper macro.
- `#define PAL_GROUP_MASK(width) ((iportmask_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_writergroup(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_ld_setpadmode`(port, pad, mode)
Pad mode setup.
- #define `pal_ld_get_pad_event`(port, pad) &_pal_events[0]; (void)(port); (void)pad
Returns a PAL event structure associated to a pad.
- #define `pal_ld_get_line_event`(line) &_pal_events[0]; (void)line
Returns a PAL event structure associated to a line.

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_PUSHPULL` 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.

PAL event modes

- #define `PAL_EVENT_MODE_EDGES_MASK` 3U
Mask of edges field.
- #define `PAL_EVENT_MODE_DISABLED` 0U
Channel disabled.
- #define `PAL_EVENT_MODE_RISING_EDGE` 1U
Rising edge callback.
- #define `PAL_EVENT_MODE_FALLING_EDGE` 2U
Falling edge callback.
- #define `PAL_EVENT_MODE_BOTH_EDGES` 3U
Both edges callback.

PAL configuration options

- #define `PAL_USE_CALLBACKS` TRUE
Enables synchronous APIs.
- #define `PAL_USE_WAIT` TRUE
Enables synchronous APIs.

Low level driver helper macros

- #define `_pal_init_event(e)`
Initializes a PAL event object.
- #define `_pal_init_event(e)`
Initializes a PAL event object.
- #define `_pal_init_event(e)`
Initializes a PAL event object.
- #define `_pal_clear_event(e)`
Clears a PAL event object.
- #define `_pal_clear_event(e)`
Clears a PAL event object.
- #define `_pal_clear_event(e)`
Clears a PAL event object.
- #define `_pal_isr_code(e)`
Common ISR code.
- #define `_pal_isr_code(e)`
Common ISR code.
- #define `_pal_isr_code(e)`
Common ISR code.

Macro Functions

- #define `palInit()` `pal_lld_init()`
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.
- `#define palEnablePadEventI(port, pad, mode)`
Pad event enable.
- `#define palDisablePadEventI(port, pad)`
Pad event disable.
- `#define palEnablePadEvent(port, pad, mode)`
Pad event enable.
- `#define palDisablePadEvent(port, pad)`
Pad event disable.
- `#define palEnableLineEventI(line, mode) palEnablePadEventI(PAL_PORT(line), PAL_PAD(line), mode)`
Line event enable.
- `#define palDisableLineEventI(line) palDisablePadEventI(PAL_PORT(line), PAL_PAD(line))`
Line event disable.
- `#define palEnableLineEvent(line, mode)`
Line event enable.
- `#define palDisableLineEvent(line)`
Line event disable.
- `#define palSetPadCallback(port, pad, cb, arg)`
Associates a callback to a pad.
- `#define palSetLineCallback(line, cb, arg)`
Associates a callback to a line.

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 void(* palcallback_t) (void *arg)`
Type of a PAL event callback.
- `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.
- `typedef uint32_t iopadid_t`
Type of an pad identifier.

Data Structures

- `struct palevent_t`
Type of a PAL event record.
- `struct IOBus`
I/O bus descriptor.
- `struct PALConfig`
Generic I/O ports static initializer.

Functions

- `ioportmask_t palReadBus (const IOBus *bus)`
Read from an I/O bus.
- `void palWriteBus (const IOBus *bus, ioportmask_t bits)`
Write to an I/O bus.
- `void palSetBusMode (const IOBus *bus, iomode_t mode)`
Programs a bus with the specified mode.
- `void palSetPadCallbackl (ioportid_t port, iopadid_t pad, palcallback_t cb, void *arg)`
Associates a callback to a port/pad.
- `void palSetLineCallbackl (ioline_t line, palcallback_t cb, void *arg)`
Associates a callback to a line.
- `msg_t palWaitPadTimeoutS (ioportid_t port, iopadid_t pad, sysinterval_t timeout)`
Waits for an edge on the specified port/pad.
- `msg_t palWaitPadTimeout (ioportid_t port, iopadid_t pad, sysinterval_t timeout)`

- Waits for an edge on the specified port/pad.*

 - `msg_t palWaitLineTimeoutS` (`ioline_t` line, `sysinterval_t` timeout)

Waits for an edge on the specified line.

 - `msg_t palWaitLineTimeout` (`ioline_t` line, `sysinterval_t` timeout)

Waits for an edge on the specified line.

 - `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.

7.39.3 Macro Definition Documentation

7.39.3.1 #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.

7.39.3.2 #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_PUSHPULL` for example.

7.39.3.3 #define PAL_MODE_INPUT 2U

Regular input high-Z pad.

7.39.3.4 #define PAL_MODE_INPUT_PULLUP 3U

Input pad with weak pull up resistor.

7.39.3.5 #define PAL_MODE_INPUT_PULLDOWN 4U

Input pad with weak pull down resistor.

7.39.3.6 #define PAL_MODE_INPUT_ANALOG 5U

Analog input mode.

7.39.3.7 #define PAL_MODE_OUTPUT_PUSHPULL 6U

Push-pull output pad.

7.39.3.8 #define PAL_MODE_OUTPUT_OPENDRAIN 7U

Open-drain output pad.

7.39.3.9 `#define PAL_LOW 0U`

Logical low state.

7.39.3.10 `#define PAL_HIGH 1U`

Logical high state.

7.39.3.11 `#define PAL_EVENT_MODE_EDGES_MASK 3U`

Mask of edges field.

7.39.3.12 `#define PAL_EVENT_MODE_DISABLED 0U`

Channel disabled.

7.39.3.13 `#define PAL_EVENT_MODE_RISING_EDGE 1U`

Rising edge callback.

7.39.3.14 `#define PAL_EVENT_MODE_FALLING_EDGE 2U`

Falling edge callback.

7.39.3.15 `#define PAL_EVENT_MODE_BOTH_EDGES 3U`

Both edges callback.

7.39.3.16 `#define PAL_USE_CALLBACKS TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.39.3.17 `#define PAL_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.39.3.18 `#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	<i>n</i>	bit position within the port
----	----------	------------------------------

Returns

The bit mask.

7.39.3.19 `#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	<i>width</i>	group width
----	--------------	-------------

Returns

The group mask.

7.39.3.20 `#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 IOBus 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

7.39.3.21 `#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 IOBus 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

7.39.3.22 `#define _pal_init_event(e)`

Value:

```
do {
    osalThreadQueueObjectInit (&_pal_events[e].threads);
    _pal_events[e].cb = NULL;
    _pal_events[e].arg = NULL;
} while (false)
```

Initializes a PAL event object.

Parameters

in	<i>e</i>	event index
----	----------	-------------

Function Class:

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

7.39.3.23 #define _pal_init_event(*e*)

Value:

```
do {
    _pal_events[e].cb = NULL;
    _pal_events[e].arg = NULL;
} while (false)
```

Initializes a PAL event object.

Parameters

in	<i>e</i>	event index
----	----------	-------------

Function Class:

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

7.39.3.24 #define _pal_init_event(*e*)

Value:

```
do {
    osalThreadQueueObjectInit (&_pal_events[e].threads);
} while (false)
```

Initializes a PAL event object.

Parameters

in	<i>e</i>	event index
----	----------	-------------

Function Class:

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

7.39.3.25 #define _pal_clear_event(e)

Value:

```
do {
   osalThreadDequeueAllI(&_pal_events[pad].threads, MSG_RESET);
    _pal_events[e].cb = NULL;
    _pal_events[e].arg = NULL;
} while (false)
```

Clears a PAL event object.

Parameters

in	<i>e</i>	event index
----	----------	-------------

Function Class:

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

7.39.3.26 #define _pal_clear_event(e)

Value:

```
do {
    _pal_events[e].cb = NULL;
    _pal_events[e].arg = NULL;
} while (false)
```

Clears a PAL event object.

Parameters

in	<i>e</i>	event index
----	----------	-------------

Function Class:

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

7.39.3.27 #define _pal_clear_event(e)

Value:

```
do {
   osalThreadDequeueAllI(&_pal_events[pad].threads, MSG_RESET);
} while (false)
```

Clears a PAL event object.

Parameters

in	<i>e</i>	event index
----	----------	-------------

Function Class:

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

7.39.3.28 `#define _pal_isr_code(e)`**Value:**

```

do {
    if ( _pal_events[e].cb != NULL) {
        _pal_events[e].cb( _pal_events[e].arg);
    }
    osalSysLockFromISR();
    osalThreadDequeueAllI (&_pal_events[e].threads, MSG_OK);
    osalSysUnlockFromISR();
} while (false)

```

Common ISR code.

Note

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

Parameters

in	<i>e</i>	event index
----	----------	-------------

Function Class:

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

7.39.3.29 `#define _pal_isr_code(e)`**Value:**

```

do {
    if ( _pal_events[e].cb != NULL) {
        _pal_events[e].cb( _pal_events[e].arg);
    }
} while (false)

```

Common ISR code.

Note

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

Parameters

in	<i>e</i>	event index
----	----------	-------------

Function Class:

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

7.39.3.30 `#define _pal_isr_code(e)`**Value:**

```

do {

```

```

osalSysLockFromISR();
osalThreadDequeueAllI(&_pal_events[e].threads, MSG_OK);
osalSysUnlockFromISR();
} while (false)

```

Common ISR code.

Note

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

Parameters

in	<i>e</i>	event index
----	----------	-------------

Function Class:

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

7.39.3.31 #define palInit() pal_Ild_init()

PAL subsystem 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.

7.39.3.32 #define palReadPort(port) ((void)(port), 0U)

Reads the physical I/O port states.

Note

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.

7.39.3.33 `#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 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.

7.39.3.34 `#define palWritePort(port, bits) ((void)(port), (void)(bits))`

Writes a bits mask on a I/O port.

Note

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.

7.39.3.35 `#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 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.

7.39.3.36 `#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 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.

7.39.3.37 `#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 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.

7.39.3.38 `#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.

7.39.3.39 `#define palWriteGroup(port, mask, offset, bits)`

Value:

```
palWritePort(port, (palReadLatch(port) & ~((mask) << (offset))) | \
                (((bits) & (mask)) << (offset)))
```

Writes a group of bits.

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 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.

7.39.3.40 `#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

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()`.
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.

7.39.3.41 `#define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1U)`

Reads an input pad logic state.

Note

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.

7.39.3.42 `#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 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 <code>PAL_LOW</code> or <code>PAL_HIGH</code>

Function Class:

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

7.39.3.43 `#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 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.

7.39.3.44 `#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 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.

7.39.3.45 `#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 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.

7.39.3.46 `#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 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()`.
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.

7.39.3.47 `#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.

7.39.3.48 `#define palWriteLine(line, bit) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)`

Writes a logic state on an output line.

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 can be called from any context.

Parameters

in	<i>line</i>	line identifier
in	<i>bit</i>	logic value, the value must be <code>PAL_LOW</code> or <code>PAL_HIGH</code>

Function Class:

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

7.39.3.49 `#define palSetLine(line) palSetPad(PAL_PORT(line), PAL_PAD(line))`

Sets a line 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 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.

7.39.3.50 `#define palClearLine(line) palClearPad(PAL_PORT(line), PAL_PAD(line))`

Clears a line 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 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.

7.39.3.51 `#define palToggleLine(line) palTogglePad(PAL_PORT(line), PAL_PAD(line))`

Toggles a line 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 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.

7.39.3.52 `#define palSetLineMode(line, mode) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)`

Line mode setup.

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 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.

7.39.3.53 #define palEnablePadEventI(port, pad, mode)

Pad event enable.

Note

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 event mode

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.

7.39.3.54 #define palDisablePadEventI(port, pad)

Pad event disable.

This function also disables previously programmed event callbacks.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

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.

7.39.3.55 #define palEnablePadEvent(port, pad, mode)**Value:**

```
do {
    osalSysLock();
    palEnablePadEventI(port, pad, mode);
    osalSysUnlock();
} while (false)
```

Pad event enable.

Note

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 event mode

Function Class:

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

7.39.3.56 #define palDisablePadEvent(port, pad)**Value:**

```
do {
    osalSysLock();
    palDisablePadEventI(port, pad);
    osalSysUnlock();
} while (false)
```

Pad event disable.

This function also disables previously programmed event callbacks.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

7.39.3.57 #define palEnableLineEventI(line, mode) palEnablePadEventI(PAL_PORT(line), PAL_PAD(line), mode)

Line event enable.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

in	<i>line</i>	line identifier
in	<i>mode</i>	line event mode

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.

7.39.3.58 `#define palDisableLineEventI(line) palDisablePadEventI(PAL_PORT(line), PAL_PAD(line))`

Line event disable.

This function also disables previously programmed event callbacks.

Parameters

in	<i>line</i>	line identifier
----	-------------	-----------------

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.

7.39.3.59 `#define palEnableLineEvent(line, mode)`

Value:

```
do {
    osalSysLock();
    palEnableLineEventI(line, mode);
    osalSysUnlock();
} while (false)
```

Line event enable.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

in	<i>line</i>	line identifier
in	<i>mode</i>	line event mode

Function Class:

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

7.39.3.60 `#define palDisableLineEvent(line)`

Value:

```
do {
    osalSysLock();
    palDisableLineEventI(line);
    osalSysUnlock();
} while (false)
```

Line event disable.

This function also disables previously programmed event callbacks.

Parameters

in	<i>line</i>	line identifier
----	-------------	-----------------

Function Class:

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

7.39.3.61 #define palSetPadCallback(port, pad, cb, arg)**Value:**

```
do {
    osalSysLock();
    palSetPadCallbackI(port, pad, cb, arg);
    osalSysUnlock();
} while (false)
```

Associates a callback to a pad.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>cb</i>	event callback function
in	<i>arg</i>	callback argument

Function Class:

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

7.39.3.62 #define palSetLineCallback(line, cb, arg)**Value:**

```
do {
    osalSysLock();
    palSetLineCallbackI(line, cb, arg);
    osalSysUnlock();
} while (false)
```

Associates a callback to a line.

Parameters

in	<i>line</i>	line identifier
in	<i>cb</i>	event callback function
in	<i>arg</i>	callback argument

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.

7.39.3.63 **#define PAL_IOPORTS_WIDTH** 16U

Width, in bits, of an I/O port.

7.39.3.64 **#define PAL_WHOLE_PORT** ((ioportmask_t)0xFFFFU)

Whole port mask.

This macro specifies all the valid bits into a port.

7.39.3.65 **#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.

7.39.3.66 **#define PAL_PORT(line)** ((stm32_gpio_t *)(((uint32_t)(line) & 0xFFFFFFF0U))

Decodes a port identifier from a line identifier.

7.39.3.67 **#define PAL_PAD(line)** ((uint32_t)((uint32_t)(line) & 0x0000000FU))

Decodes a pad identifier from a line identifier.

7.39.3.68 **#define PAL_NOLINE** 0U

Value identifying an invalid line.

7.39.3.69 **#define IOPORT1** 0

First I/O port identifier.

Low level drivers can define multiple ports, it is suggested to use this naming convention.

7.39.3.70 **#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.

7.39.3.71 **#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.

7.39.3.72 #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.

7.39.3.73 #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.

7.39.3.74 #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.

7.39.3.75 `#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.

7.39.3.76 `#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.

7.39.3.77 #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.

7.39.3.78 #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.

7.39.3.79 `#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.

7.39.3.80 `#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.

7.39.3.81 `#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 <code>PAL_LOW</code> or <code>PAL_HIGH</code>

Function Class:

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

7.39.3.82 `#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.

7.39.3.83 #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.

7.39.3.84 #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.

7.39.3.85 `#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.

7.39.3.86 `#define pal_lld_get_pad_event(port, pad) &_pal_events[0]; (void)(port); (void)pad`

Returns a PAL event structure associated to a pad.

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.

7.39.3.87 `#define pal_lld_get_line_event(line) &_pal_events[0]; (void)line`

Returns a PAL event structure associated to a line.

Parameters

in	<i>line</i>	line identifier
----	-------------	-----------------

Function Class:

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

7.39.4 Typedef Documentation**7.39.4.1 typedef void(* palcallback_t) (void *arg)**

Type of a PAL event callback.

7.39.4.2 typedef uint32_t ioportmask_t

Digital I/O port sized unsigned type.

7.39.4.3 typedef uint32_t iomode_t

Digital I/O modes.

7.39.4.4 typedef uint32_t ioline_t

Type of an I/O line.

7.39.4.5 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.

7.39.4.6 typedef uint32_t iopadid_t

Type of an pad identifier.

7.39.5 Function Documentation**7.39.5.1 ioportmask_t palReadBus (const 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	bus	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.

7.39.5.2 void palWriteBus (const 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 IOBus 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.

7.39.5.3 void palSetBusMode (const 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 IOBus structure
in	<i>mode</i>	the mode

Function Class:

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

7.39.5.4 `void palSetPadCallback(ioportid_t port, iopadid_t pad, palcallback_t cb, void * arg)`

Associates a callback to a port/pad.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>cb</i>	event callback function
in	<i>arg</i>	callback argument

Function Class:

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

7.39.5.5 `void palSetLineCallback(ioline_t line, palcallback_t cb, void * arg)`

Associates a callback to a line.

Parameters

in	<i>line</i>	line identifier
in	<i>cb</i>	event callback function
in	<i>arg</i>	callback argument

Function Class:

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

7.39.5.6 `msg_t palWaitPadTimeoutS(ioportid_t port, iopadid_t pad, sysinterval_t timeout)`

Waits for an edge on the specified port/pad.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation state.

Return values

<i>MSG_OK</i>	if an edge has been detected.
<i>MSG_TIMEOUT</i>	if a timeout occurred before an edge could be detected.
<i>MSG_RESET</i>	if the event has been disabled while the thread was waiting for an edge.

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:



7.39.5.7 msg_t palWaitPadTimeout (ioportid_t port, iopadid_t pad, sysinterval_t timeout)

Waits for an edge on the specified port/pad.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation state.

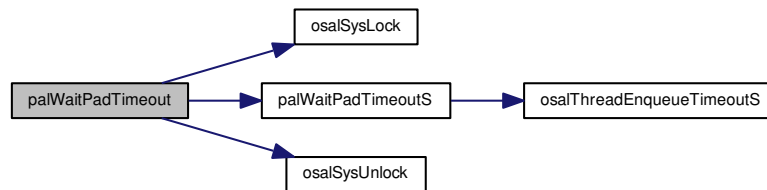
Return values

<i>MSG_OK</i>	if an edge has been detected.
<i>MSG_TIMEOUT</i>	if a timeout occurred before an edge could be detected.
<i>MSG_RESET</i>	if the event has been disabled while the thread was waiting for an edge.

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:



7.39.5.8 msg_t palWaitLineTimeoutS (ioline_t line, sysinterval_t timeout)

Waits for an edge on the specified line.

Parameters

in	<i>line</i>	line identifier
in	<i>timeout</i>	operation timeout

Returns

The operation state.

Return values

<i>MSG_OK</i>	if an edge has been detected.
<i>MSG_TIMEOUT</i>	if a timeout occurred before an edge could be detected.
<i>MSG_RESET</i>	if the event has been disabled while the thread was waiting for an edge.

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:



7.39.5.9 msg_t palWaitLineTimeout (ioline_t line, sysinterval_t timeout)

Waits for an edge on the specified line.

Parameters

in	<i>line</i>	line identifier
in	<i>timeout</i>	operation timeout

Returns

The operation state.

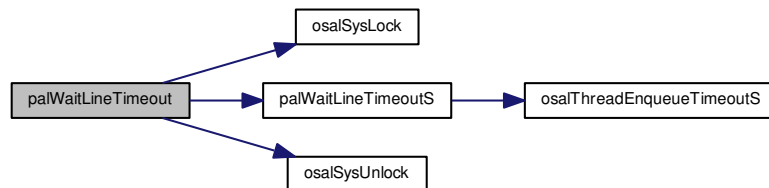
Return values

<i>MSG_OK</i>	if an edge has been detected.
<i>MSG_TIMEOUT</i>	if a timeout occurred before an edge could be detected.
<i>MSG_RESET</i>	if the event has been disabled while the thread was waiting for an edge.

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:

**7.39.5.10 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.

7.39.5.11 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.

7.40 PWM Driver

Generic PWM Driver.

7.40.1 Detailed Description

Generic PWM Driver.

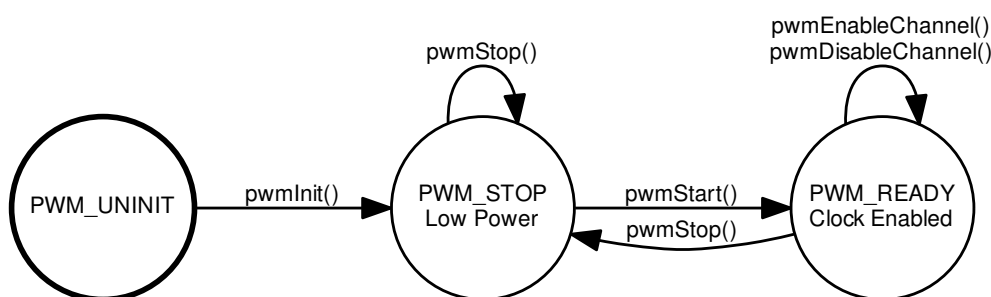
This module implements a generic PWM (Pulse Width Modulation) driver.

Precondition

In order to use the PWM driver the `HAL_USE_PWM` option must be enabled in `halconf.h`.

7.40.2 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).



7.40.3 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 `pwmIsChannelEnabledI`(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 `pwmmode_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.

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, `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.
- 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`)

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

Variables

- `PWMDriver PWMD1`
PWMD1 driver identifier.

7.40.4 Macro Definition Documentation

7.40.4.1 #define PWM_OUTPUT_MASK 0x0FU

Standard output modes mask.

7.40.4.2 #define PWM_OUTPUT_DISABLED 0x00U

Output not driven, callback only.

7.40.4.3 #define PWM_OUTPUT_ACTIVE_HIGH 0x01U

Positive PWM logic, active is logic level one.

7.40.4.4 #define PWM_OUTPUT_ACTIVE_LOW 0x02U

Inverse PWM logic, active is logic level zero.

7.40.4.5 #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 PWMDriver 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.

7.40.4.6 #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 PWMDriver 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.

7.40.4.7 `#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 PWMDriver 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.

7.40.4.8 `#define pwmChangePeriod(pwmp, value)`

Value:

```
{
    (pwmp)->period = (value);
    pwm\_llc\_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 PWMDriver 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.

7.40.4.9 #define pwmEnableChannel(pwmp, channel, width)**Value:**

```
do {
    (pwmp)->enabled |= ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel));
    pwm_llc_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.

7.40.4.10 #define pwmDisableChannel(pwmp, channel)**Value:**

```
do {
    (pwmp)->enabled &= ~((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel));
    pwm_llc_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 PWMDriver 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.

```
7.40.4.11 #define pwmlsChannelEnabled( 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 PWMDriver 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.

```
7.40.4.12 #define pwmEnablePeriodicNotification( pwmp ) pwm_ild_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 PWMDriver 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.

7.40.4.13 `#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 PWMDriver 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.

7.40.4.14 `#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 PWMDriver 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.

7.40.4.15 `#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 <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.

7.40.4.16 `#define PWM_CHANNELS 4`

Number of PWM channels per PWM driver.

7.40.4.17 `#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`.

7.40.4.18 `#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 PWMDriver object
in	<i>period</i>	new cycle time in ticks

Function Class:

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

7.40.5 Typedef Documentation**7.40.5.1 typedef struct PWMDriver PWMDriver**

Type of a structure representing a PWM driver.

7.40.5.2 typedef void(* pwmcallback_t) (PWMDriver *pwmp)

Type of a PWM notification callback.

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
----	-------------	---

7.40.5.3 typedef uint32_t pwmmode_t

Type of a PWM mode.

7.40.5.4 typedef uint8_t pwmchannel_t

Type of a PWM channel.

7.40.5.5 typedef uint32_t pwmchnmsk_t

Type of a channels mask.

7.40.5.6 typedef uint32_t pwmcnt_t

Type of a PWM counter.

7.40.6 Enumeration Type Documentation

7.40.6.1 enum pwmstate_t

Driver state machine possible states.

Enumerator

PWM_UNINIT Not initialized.

PWM_STOP Stopped.

PWM_READY Ready.

7.40.7 Function Documentation

7.40.7.1 void pwmInit (void)

PWM Driver initialization.

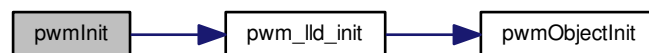
Note

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:



7.40.7.2 void pwmObjectInit (PWMDriver * pwmp)

Initializes the standard part of a `PWMDriver` structure.

Parameters

out	<i>pwmp</i>	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.

7.40.7.3 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.

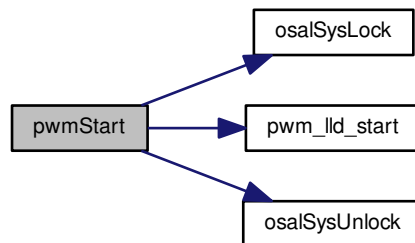
Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>config</i>	pointer to a PWMConfig 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:

**7.40.7.4 void pwmStop (PWMDriver * pwmp)**

Deactivates the PWM peripheral.

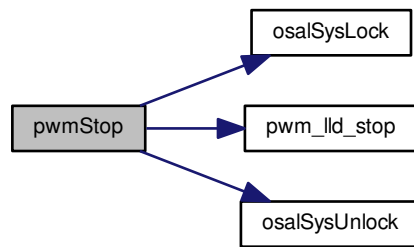
Parameters

in	<i>pwmp</i>	pointer to a PWMDriver 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:



7.40.7.5 void pwmChangePeriod (PWMDriver * pwmp, pwmcnt_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.

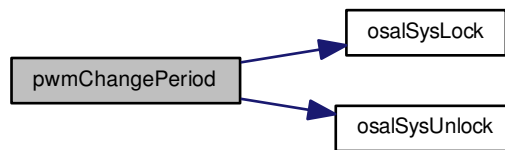
Parameters

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:



7.40.7.6 void pwmEnableChannel (PWMDriver * *pwmp*, pwmchannel_t *channel*, pwmcnt_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).

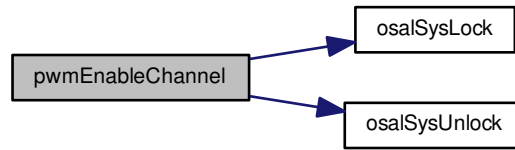
Parameters

in	<i>pwmp</i>	pointer to a PWMDriver 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:



7.40.7.7 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).

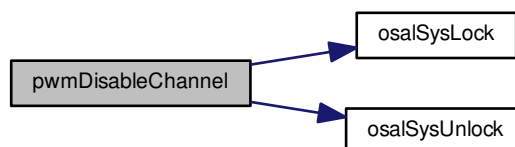
Parameters

in	<i>pwmp</i>	pointer to a PWMDriver 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:



7.40.7.8 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.

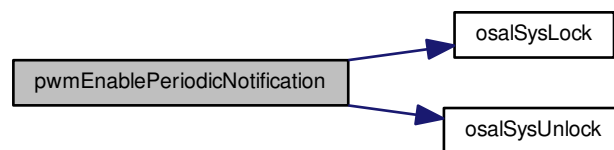
Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</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:



7.40.7.9 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.

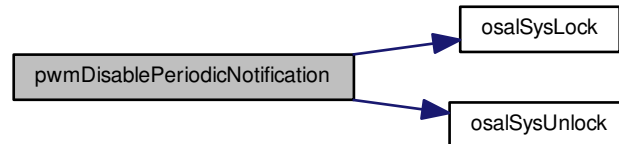
Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</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:



7.40.7.10 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.

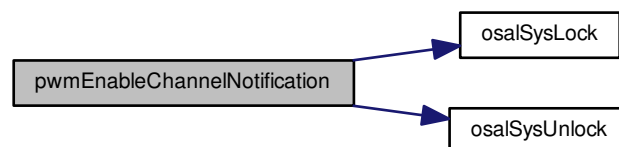
Parameters

in	<i>pwmp</i>	pointer to a PWMDriver 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:



7.40.7.11 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.

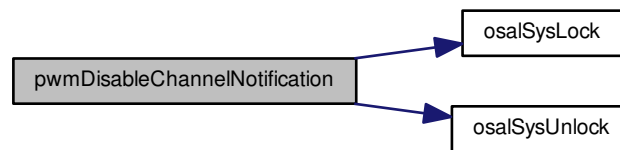
Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> 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:



7.40.7.12 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:



7.40.7.13 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	<i>pwmp</i>	pointer to a PWMDriver object
----	-------------	---

Function Class:

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

7.40.7.14 void pwm_lld_stop (PWMDriver * pwmp)

Deactivates the PWM peripheral.

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
----	-------------	---

Function Class:

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

7.40.7.15 void pwm_lld_enable_channel (PWMDriver * pwmp, pwmchannel_t channel, pwmcnt_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 PWMDriver 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.

7.40.7.16 `void pwm_lld_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 PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

7.40.7.17 `void pwm_lld_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 PWMDriver object
----	-------------	---

Function Class:

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

7.40.7.18 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 <code>PWMDriver</code> object
----	-------------	--

Function Class:

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

7.40.7.19 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 <code>PWMDriver</code> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

7.40.7.20 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 PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

7.40.8 Variable Documentation**7.40.8.1 PWMDriver PWMD1**

PWMD1 driver identifier.

Note

The driver PWMD1 allocates the complex timer TIM1 when enabled.

7.41 QSPI Driver

Generic QSPI Driver.

7.41.1 Detailed Description

Generic QSPI Driver.

This module defines an abstract interface for a Quad SPI communication interface.

Precondition

In order to use the QSPI driver the `HAL_USE_QSPI` option must be enabled in `halconf.h`.

Transfer options

- `#define QSPI_CFG_CMD_MASK` (0xFFLU << 0LU)
- `#define QSPI_CFG_CMD(n)` ((n) << 0LU)
- `#define QSPI_CFG_CMD_MODE_MASK` (3LU << 8LU)
- `#define QSPI_CFG_CMD_MODE_NONE` (0LU << 8LU)
- `#define QSPI_CFG_CMD_MODE_ONE_LINE` (1LU << 8LU)
- `#define QSPI_CFG_CMD_MODE_TWO_LINES` (2LU << 8LU)
- `#define QSPI_CFG_CMD_MODE_FOUR_LINES` (3LU << 8LU)
- `#define QSPI_CFG_ADDR_MODE_MASK` (3LU << 10LU)
- `#define QSPI_CFG_ADDR_MODE_NONE` (0LU << 10LU)
- `#define QSPI_CFG_ADDR_MODE_ONE_LINE` (1LU << 10LU)
- `#define QSPI_CFG_ADDR_MODE_TWO_LINES` (2LU << 10LU)
- `#define QSPI_CFG_ADDR_MODE_FOUR_LINES` (3LU << 10LU)
- `#define QSPI_CFG_ADDR_SIZE_MASK` (3LU << 12LU)
- `#define QSPI_CFG_ADDR_SIZE_8` (0LU << 12LU)
- `#define QSPI_CFG_ADDR_SIZE_16` (1LU << 12LU)
- `#define QSPI_CFG_ADDR_SIZE_24` (2LU << 12LU)
- `#define QSPI_CFG_ADDR_SIZE_32` (3LU << 12LU)
- `#define QSPI_CFG_ALT_MODE_MASK` (3LU << 14LU)
- `#define QSPI_CFG_ALT_MODE_NONE` (0LU << 14LU)
- `#define QSPI_CFG_ALT_MODE_ONE_LINE` (1LU << 14LU)
- `#define QSPI_CFG_ALT_MODE_TWO_LINES` (2LU << 14LU)
- `#define QSPI_CFG_ALT_MODE_FOUR_LINES` (3LU << 14LU)
- `#define QSPI_CFG_ALT_SIZE_MASK` (3LU << 16LU)
- `#define QSPI_CFG_ALT_SIZE_8` (0LU << 16LU)
- `#define QSPI_CFG_ALT_SIZE_16` (1LU << 16LU)
- `#define QSPI_CFG_ALT_SIZE_24` (2LU << 16LU)
- `#define QSPI_CFG_ALT_SIZE_32` (3LU << 16LU)
- `#define QSPI_CFG_DUMMY_CYCLES_MASK` (0x1FLU << 18LU)
- `#define QSPI_CFG_DUMMY_CYCLES(n)` ((n) << 18LU)
- `#define QSPI_CFG_DATA_MODE_MASK` (3LU << 24LU)
- `#define QSPI_CFG_DATA_MODE_NONE` (0LU << 24LU)
- `#define QSPI_CFG_DATA_MODE_ONE_LINE` (1LU << 24LU)
- `#define QSPI_CFG_DATA_MODE_TWO_LINES` (2LU << 24LU)
- `#define QSPI_CFG_DATA_MODE_FOUR_LINES` (3LU << 24LU)
- `#define QSPI_CFG_SIOO` (1LU << 28LU)
- `#define QSPI_CFG_DDRM` (1LU << 31LU)

QSPI configuration options

- #define `QSPI_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `QSPI_USE_MUTUAL_EXCLUSION` TRUE
Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Macro Functions

- #define `qspiStartCommandl`(qspi, cmdp)
Sends a command without data phase.
- #define `qspiStartSendl`(qspi, cmdp, n, txbuf)
Sends data over the QSPI bus.
- #define `qspiStartReceivel`(qspi, cmdp, n, rxbuf)
Receives data from the QSPI bus.
- #define `qspiMapFlashl`(qspi, cmdp, addrp) `qspi_llc_map_flash`(qspi, cmdp, addrp)
Maps in memory space a QSPI flash device.
- #define `qspiUnmapFlashl`(qspi) `qspi_llc_unmap_flash`(qspi)
Maps in memory space a QSPI flash device.

Low level driver helper macros

- #define `_qspi_wakeup_isr`(qspi)
Wakes up the waiting thread.
- #define `_qspi_isr_code`(qspi)
Common ISR code.

QSPI capabilities

- #define `QSPI_SUPPORTS_MEMMAP` TRUE

Configuration options

- #define `PLATFORM_QSPI_USE_QSPI1` FALSE
QSPID1 driver enable switch.

Typedefs

- typedef struct `QSPIDriver` `QSPIDriver`
Type of a structure representing an QSPI driver.
- typedef void(* `qspicallback_t`) (`QSPIDriver` *qspi)
Type of a QSPI notification callback.

Data Structures

- struct `qspi_command_t`
Type of a QSPI command descriptor.
- struct `QSPIDriver`
Driver configuration structure.
- struct `QSPIDriver`
Structure representing an QSPI driver.

Functions

- void **qsplInit** (void)
QSPI Driver initialization.
- void **qsplObjectInit** (QSPIDriver *qspip)
*Initializes the standard part of a *QSPIDriver* structure.*
- void **qsplStart** (QSPIDriver *qspip, const QSPIConfig *config)
Configures and activates the QSPI peripheral.
- void **qsplStop** (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void **qsplStartCommand** (QSPIDriver *qspip, const qspl_command_t *cmdp)
Sends a command without data phase.
- void **qsplStartSend** (QSPIDriver *qspip, const qspl_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void **qsplStartReceive** (QSPIDriver *qspip, const qspl_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void **qsplCommand** (QSPIDriver *qspip, const qspl_command_t *cmdp)
Sends a command without data phase.
- void **qsplSend** (QSPIDriver *qspip, const qspl_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void **qsplReceive** (QSPIDriver *qspip, const qspl_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void **qsplMapFlash** (QSPIDriver *qspip, const qspl_command_t *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void **qsplUnmapFlash** (QSPIDriver *qspip)
Unmaps from memory space a QSPI flash device.
- void **qsplAcquireBus** (QSPIDriver *qspip)
Gains exclusive access to the QSPI bus.
- void **qsplReleaseBus** (QSPIDriver *qspip)
Releases exclusive access to the QSPI bus.
- void **qspl_lld_init** (void)
Low level QSPI driver initialization.
- void **qspl_lld_start** (QSPIDriver *qspip)
Configures and activates the QSPI peripheral.
- void **qspl_lld_stop** (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void **qspl_lld_command** (QSPIDriver *qspip, const qspl_command_t *cmdp)
Sends a command without data phase.
- void **qspl_lld_send** (QSPIDriver *qspip, const qspl_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void **qspl_lld_receive** (QSPIDriver *qspip, const qspl_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void **qspl_lld_map_flash** (QSPIDriver *qspip, const qspl_command_t *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void **qspl_lld_unmap_flash** (QSPIDriver *qspip)
Unmaps from memory space a QSPI flash device.

Enumerations

Variables

- **QSPIDriver QSPID1**
QSPID1 driver identifier.

7.41.2 Macro Definition Documentation

7.41.2.1 `#define QSPI_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.41.2.2 `#define QSPI_USE_MUTUAL_EXCLUSION TRUE`

Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.41.2.3 `#define qspiStartCommand(qspip, cmdp)`

Value:

```
{
    osalDbgAssert(((cmdp)->cfg & QSPI_CFG_DATA_MODE_MASK) ==
                  QSPI_CFG_DATA_MODE_NONE,
                  "data mode specified");
    (qspip)->state = QSPI_ACTIVE;
    qspi_llc_command(qspip, cmdp);
}
```

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<code>qspip</code>	pointer to the <code>QSPIDriver</code> object
in	<code>cmdp</code>	pointer to the command descriptor

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.

7.41.2.4 `#define qspiStartSend(qspip, cmdp, n, txbuf)`

Value:

```
{
    osalDbgAssert(((cmdp)->cfg & QSPI_CFG_DATA_MODE_MASK) !=
                  QSPI_CFG_DATA_MODE_NONE,
                  "data mode required");
    (qspip)->state = QSPI_ACTIVE;
}
```

```

    qspi_llt_send(qspip, cmdp, n, txbuf);
}

```

Sends data over the QSPI bus.

This asynchronous function starts a transmit operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send or zero if no data phase
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.

7.41.2.5 #define qspiStartReceive(*qspip, cmdp, n, rxbuf*)

Value:

```

{
    osalDbgAssert(((cmdp)->cfg & QSPI_CFG_DATA_MODE_MASK) !=
        QSPI_CFG_DATA_MODE_NONE,
        "data mode required");
    (qspip)->state = QSPI_ACTIVE;
    qspi_llt_receive(qspip, cmdp, n, rxbuf);
}

```

Receives data from the QSPI bus.

This asynchronous function starts a receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to receive or zero if no data phase
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.

7.41.2.6 `#define qspiMapFlash(qspip, cmdp, addrp) qspi_ild_map_flash(qspip, cmdp, addrp)`

Maps in memory space a QSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
out	<i>addrp</i>	pointer to the memory start address of the mapped flash or <code>NULL</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.

7.41.2.7 `#define qspiUnmapFlash(qspip) qspi_ild_unmap_flash(qspip)`

Maps in memory space a QSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver 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.

7.41.2.8 `#define _qspi_wakeup_isr(qspip)`

Value:

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

Wakes up the waiting thread.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
----	--------------	--

Function Class:

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

7.41.2.9 #define _qspi_isr_code(qspip)**Value:**

```

{
    if ((qspip)->config->end_cb) {
        (qspip)->state = QSPI_COMPLETE;
        (qspip)->config->end_cb(qspip);
        if ((qspip)->state == QSPI_COMPLETE)
            (qspip)->state = QSPI_READY;
    }
    else
        (qspip)->state = QSPI_READY;
    _qspi_wakeup_isr(qspip);
}

```

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>qspip</i>	pointer to the QSPIDriver object
----	--------------	--

Function Class:

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

7.41.2.10 #define PLATFORM_QSPI_USE_QSPID1 FALSE

QSPID1 driver enable switch.

If set to `TRUE` the support for QSPID1 is included.

Note

The default is `FALSE`.

7.41.3 Typedef Documentation**7.41.3.1 typedef struct QSPIDriver QSPIDriver**

Type of a structure representing an QSPI driver.

7.41.3.2 typedef void(* qspicallback_t) (QSPIDriver *qspip)

Type of a QSPI notification callback.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object triggering the callback
----	--------------	--

7.41.4 Enumeration Type Documentation

7.41.4.1 enum qspistate_t

Driver state machine possible states.

Enumerator

- QSPI_UNINIT** Not initialized.
- QSPI_STOP** Stopped.
- QSPI_READY** Ready.
- QSPI_ACTIVE** Exchanging data.
- QSPI_COMPLETE** Asynchronous operation complete.
- QSPI_MEMMAP** In memory mapped mode.

7.41.5 Function Documentation

7.41.5.1 void qspilnit (void)

QSPI Driver initialization.

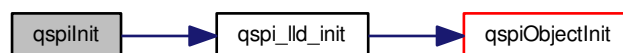
Note

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:



7.41.5.2 void qspiObjectInit (QSPIDriver * qspip)

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

Parameters

out	<i>qspip</i>	pointer to the QSPIDriver 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:

**7.41.5.3 void qspiStart (QSPIDriver * *qspip*, const QSPICConfig * *config*)**

Configures and activates the QSPI peripheral.

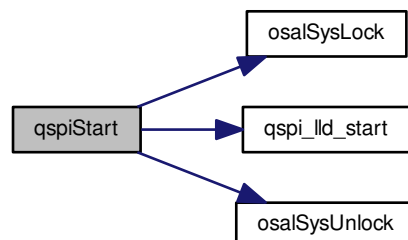
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>config</i>	pointer to the QSPICConfig 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:



7.41.5.4 void qspiStop (QSPIDriver * *qspip*)

Deactivates the QSPI peripheral.

Note

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

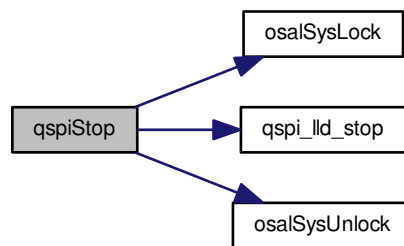
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver 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:



7.41.5.5 void qspiStartCommand (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*)

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

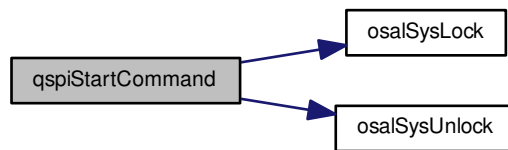
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor

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:



7.41.5.6 void qspiStartSend (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*, size_t *n*, const uint8_t * *txbuf*)

Sends a command with data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

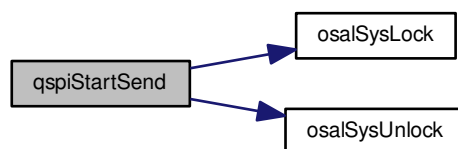
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes 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:



7.41.5.7 void qspiStartReceive (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*, size_t *n*, uint8_t * *rxbuf*)

Sends a command then receives data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

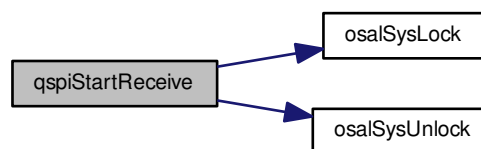
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send
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:



7.41.5.8 void qspiCommand (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*)

Sends a command without data phase.

Precondition

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

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

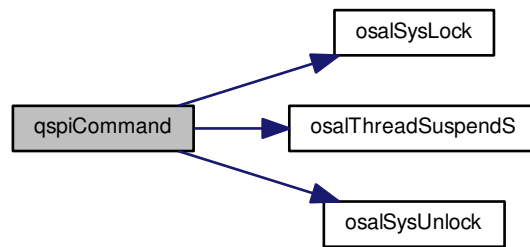
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor

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:



7.41.5.9 `void qspiSend (QSPIDriver * qspip, const qspi_command_t * cmdp, size_t n, const uint8_t * txbuf)`

Sends a command with data over the QSPI bus.

Precondition

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

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

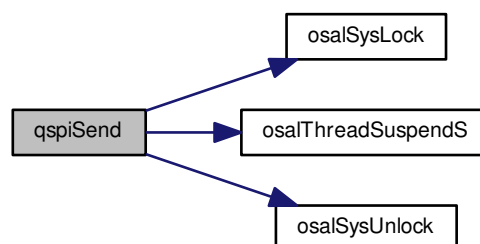
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes 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:



7.41.5.10 void qspiReceive (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*, size_t *n*, uint8_t * *rxbuf*)

Sends a command then receives data over the QSPI bus.

Precondition

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

In order to use this function the driver must have been configured without callbacks (*end_cb* = NULL).

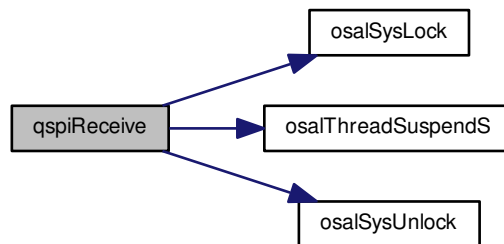
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send
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:



7.41.5.11 void qspiMapFlash (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*, uint8_t ** *addrp*)

Maps in memory space a QSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

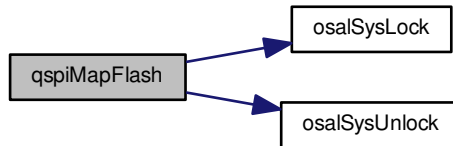
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
out	<i>addrp</i>	pointer to the memory start address of the mapped flash or 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:



7.41.5.12 void qspiUnmapFlash (QSPIDriver * *qspip*)

Unmaps from memory space a QSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

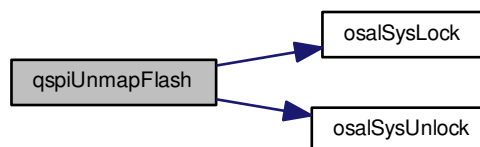
Parameters

in	<i>qspip</i>	pointer to the QSPIDriver 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:



7.41.5.13 void qspiAcquireBus (QSPIDriver * *qspip*)

Gains exclusive access to the QSPI bus.

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

Precondition

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

Parameters

in	<i>qspi</i>	pointer to the QSPIDriver 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:



7.41.5.14 void qspiReleaseBus (QSPIDriver * *qspi*)

Releases exclusive access to the QSPI bus.

Precondition

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

Parameters

in	<i>qspi</i>	pointer to the QSPIDriver 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:



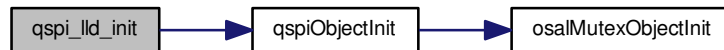
7.41.5.15 void qspi_lld_init (void)

Low level QSPI driver initialization.

Function Class:

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

Here is the call graph for this function:



7.41.5.16 void qspi_lld_start (QSPIDriver * qspip)

Configures and activates the QSPI peripheral.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
----	--------------	--

Function Class:

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

7.41.5.17 void qspi_lld_stop (QSPIDriver * qspip)

Deactivates the QSPI peripheral.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
----	--------------	--

Function Class:

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

7.41.5.18 void qspi_lld_command (QSPIDriver * qspip, const qspi_command_t * cmdp)

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor

Function Class:

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

7.41.5.19 void `qspi_llid_send (QSPIDriver * qspip, const qspi_command_t * cmdp, size_t n, const uint8_t * txbuf)`

Sends a command with data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send
in	<i>txbuf</i>	the pointer to the transmit buffer

Function Class:

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

7.41.5.20 void `qspi_llid_receive (QSPIDriver * qspip, const qspi_command_t * cmdp, size_t n, uint8_t * rxbuf)`

Sends a command then receives data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

7.41.5.21 void `qspi_llid_map_flash (QSPIDriver * qspip, const qspi_command_t * cmdp, uint8_t ** addrp)`

Maps in memory space a QSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
out	<i>addrp</i>	pointer to the memory start address of the mapped flash or <code>NULL</code>

Function Class:

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

7.41.5.22 void qspi_lld_unmap_flash (QSPIDriver * *qspip*)

Unmaps from memory space a QSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
----	--------------	--

Function Class:

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

7.41.6 Variable Documentation**7.41.6.1 QSPIDriver QSPID1**

QSPID1 driver identifier.

7.42 RTC Driver

Generic RTC Driver.

7.42.1 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 in 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 `RTCAAlarm`
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 `RTCAAlarm` *alarmspec)
Set alarm time.
- void `rtcGetAlarm` (`RTCDriver` *rtcp, `rtcalarm_t` alarm, `RTCAAlarm` *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 `rtcConvertStructTmToDateTime` (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)
RTC driver identifier.
- 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 `RTCAAlarm` *alarmspec)

Set alarm time.

- void `rtc_lld_get_alarm` (`RTCDriver` *rtcp, `rtcalarm_t` alarm, `RTCAAlarm` *alarmspec)

Get alarm time.

Enumerations

7.42.2 Macro Definition Documentation

7.42.2.1 #define RTC_BASE_YEAR 1980U

Base year of the calendar.

7.42.2.2 #define RTC_SUPPORTS_CALLBACKS TRUE

Callback support in the driver.

7.42.2.3 #define RTC_ALARMS 2

Number of alarms available.

7.42.2.4 #define RTC_HAS_STORAGE FALSE

Presence of a local persistent storage.

7.42.2.5 #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`.

7.42.2.6 #define _rtc_driver_methods _file_stream_methods

`FileStream` specific methods.

7.42.3 Typedef Documentation

7.42.3.1 typedef struct RTCDriver RTCDriver

Type of a structure representing an RTC driver.

7.42.3.2 typedef uint32_t rtcalarm_t

Type of an RTC alarm number.

7.42.3.3 `typedef void(* rtccb_t) (RTCDriver *rtcp, rtcevent_t event)`

Type of a generic RTC callback.

7.42.4 Enumeration Type Documentation

7.42.4.1 `enum rtcevent_t`

Type of an RTC event.

7.42.5 Function Documentation

7.42.5.1 `void rtcInit (void)`

RTC Driver initialization.

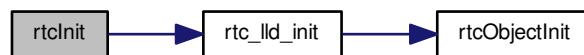
Note

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:



7.42.5.2 `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.

7.42.5.3 `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 RTCDateTime structure

Function Class:

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

Here is the call graph for this function:

**7.42.5.4 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 RTCDateTime structure

Function Class:

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

Here is the call graph for this function:



7.42.5.5 void rtcSetAlarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, const RTCAAlarm * *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 RTCAAlarm structure or NULL

Function Class:

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

Here is the call graph for this function:



7.42.5.6 void rtcGetAlarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, RTCAAlarm * *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 RTCAlarm structure

Function Class:

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

Here is the call graph for this function:

**7.42.5.7 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.

7.42.5.8 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 RTCDateTime structure
out	<i>tmp</i>	pointer to a broken-down time structure
out	<i>tv_msec</i>	pointer to milliseconds value or <code>NULL</code>

Function Class:

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

7.42.5.9 `void rtcConvertStructTmToDateTime (const struct tm * timep, uint32_t tv_nsec, RTCDatetime * timespec)`

Convert broken-down time structure to [RTCDatetime](#).

Parameters

in	<i>timep</i>	pointer to a broken-down time structure
in	<i>tv_nsec</i>	nanoseconds value
out	<i>timespec</i>	pointer to a RTCDatetime structure

Function Class:

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

7.42.5.10 `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 RTCDatetime 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.

7.42.5.11 `void rtc_lld_init (void)`

RTC driver identifier.

Enable access to registers.

Function Class:

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

Here is the call graph for this function:



7.42.5.12 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 RTCDateTime structure

Function Class:

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

7.42.5.13 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 RTCDateTime structure

Function Class:

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

7.42.5.14 void rtc_lld_set_alarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, const RTCAAlarm * *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 RTCAAlarm structure.

Function Class:

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

7.42.5.15 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 RTCAAlarm structure

Function Class:

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

7.43 SDC Driver

Generic SD Card Driver.

7.43.1 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`.

7.43.2 Driver State Machine

This driver implements a state machine internally, see the [Abstract I/O Block Device](#) module documentation for details.

7.43.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Macros

- `#define _sdc_driver_methods _mmcscd_block_device_methods`
SDCDriver 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.
- #define `SDC_INIT_OCR_V20` 0x50FF8000U
OCR initialization constant for V20 cards.
- #define `SDC_INIT_OCR` 0x80100000U
OCR initialization constant for non-V20 cards.

Macro Functions

- #define `sdclsCardInserted`(sdcp) (`sdclld_is_card_inserted`(sdcp))
Returns the card insertion status.
- #define `sdclsWriteProtected`(sdcp) (`sdclld_is_write_protected`(sdcp))
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 `sdclflags_t`
SDC Driver condition flags type.
- typedef struct `SDCDriver` `SDCDriver`
Type of a structure representing an SDC driver.

Data Structures

- struct `SDCConfig`
Driver configuration structure.
- struct `SDCDriverVMT`
`SDCDriver` virtual methods table.
- struct `SDCDriver`
Structure representing an SDC driver.

Functions

- static bool `mode_detect` (`SDCDriver` *sdcp)
Detects card mode.
- static bool `mmc_init` (`SDCDriver` *sdcp)
Init procedure for MMC.
- static bool `sdcl_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 `sd_cmd6_construct` (`sd_switch_t` mode, `sd_switch_function_t` function, uint32_t value)

Constructs CMD6 argument for SDC.
- static uint16_t `sd_cmd6_extract_info` (`sd_switch_function_t` function, const uint8_t *buf)

Extracts information from CMD6 answer.
- static bool `sd_cmd6_check_status` (`sd_switch_function_t` function, const uint8_t *buf)

Checks status after switching using CMD6.
- static bool `sd_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 `sd_set_bus_width` (`SDCDriver` *sdcp)

Sets bus width for SDC.
- static bool `mmc_set_bus_width` (`SDCDriver` *sdcp)

Sets bus width for MMC.
- bool `_sd_wait_for_transfer_state` (`SDCDriver` *sdcp)

Wait for the card to complete pending operations.
- void `sdclnit` (void)

SDC Driver initialization.
- void `sdObjectInit` (`SDCDriver` *sdcp)

Initializes the standard part of a `SDCDriver` structure.
- void `sdStart` (`SDCDriver` *sdcp, const `SDCConfig` *config)

Configures and activates the SDC peripheral.
- void `sdStop` (`SDCDriver` *sdcp)

Deactivates the SDC peripheral.
- bool `sdConnect` (`SDCDriver` *sdcp)

Performs the initialization procedure on the inserted card.
- bool `sdDisconnect` (`SDCDriver` *sdcp)

Brings the driver in a state safe for card removal.
- bool `sdRead` (`SDCDriver` *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)

Reads one or more blocks.
- bool `sdWrite` (`SDCDriver` *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)

Writes one or more blocks.
- `sdclflags_t` `sdGetAndClearErrors` (`SDCDriver` *sdcp)

Returns the errors mask associated to the previous operation.
- bool `sdSync` (`SDCDriver` *sdcp)

Waits for card idle condition.
- bool `sdGetInfo` (`SDCDriver` *sdcp, `BlockDeviceInfo` *bdip)

Returns the media info.
- bool `sdErase` (`SDCDriver` *sdcp, uint32_t startblk, uint32_t endblk)

Erases the supplied blocks.
- void `sd_llc_init` (void)

Low level SDC driver initialization.
- void `sd_llc_start` (`SDCDriver` *sdcp)

Configures and activates the SDC peripheral.
- void `sd_llc_stop` (`SDCDriver` *sdcp)

Deactivates the SDC peripheral.

- void `sd_c_lld_start_clk` (`SDCDriver *sdcp`)
Starts the SDIO clock and sets it to init mode (400kHz or less).
- void `sd_c_lld_set_data_clk` (`SDCDriver *sdcp`, `sdcbusclk_t clk`)
Sets the SDIO clock to data mode (25MHz or less).
- void `sd_c_lld_stop_clk` (`SDCDriver *sdcp`)
Stops the SDIO clock.
- void `sd_c_lld_set_bus_mode` (`SDCDriver *sdcp`, `sdcbusmode_t mode`)
Switches the bus to 4 bits mode.
- void `sd_c_lld_send_cmd_none` (`SDCDriver *sdcp`, `uint8_t cmd`, `uint32_t arg`)
Sends an SDIO command with no response expected.
- bool `sd_c_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 `sd_c_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 `sd_c_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 `sd_c_lld_read` (`SDCDriver *sdcp`, `uint32_t startblk`, `uint8_t *buf`, `uint32_t n`)
Reads one or more blocks.
- bool `sd_c_lld_write` (`SDCDriver *sdcp`, `uint32_t startblk`, `const uint8_t *buf`, `uint32_t n`)
Writes one or more blocks.
- bool `sd_c_lld_sync` (`SDCDriver *sdcp`)
Waits for card idle condition.

Enumerations

Variables

- static const struct `SDCDriverVMT sd_c_vmt`
Virtual methods table.
- `SDCDriver SDCD1`
SDCD1 driver identifier.

7.43.4 Macro Definition Documentation

7.43.4.1 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.43.4.2 #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.

7.43.4.3 `#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.

7.43.4.4 `#define SDC_INIT_OCR_V20 0x50FF8000U`

OCR initialization constant for V20 cards.

7.43.4.5 `#define SDC_INIT_OCR 0x80100000U`

OCR initialization constant for non-V20 cards.

7.43.4.6 `#define sdclsCardInserted(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 SDCDriver 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.

7.43.4.7 `#define sdclsWriteProtected(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	<i>sdcp</i>	pointer to the SDCDriver 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.

7.43.4.8 `#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`.

7.43.4.9 `#define _sdc_driver_methods _mmc_sdc_block_device_methods`

[SDCDriver](#) specific methods.

7.43.5 Typedef Documentation

7.43.5.1 `typedef uint32_t sdcmode_t`

Type of card flags.

7.43.5.2 `typedef uint32_t sdcflags_t`

SDC Driver condition flags type.

7.43.5.3 `typedef struct SDCDriver SDCDriver`

Type of a structure representing an SDC driver.

7.43.6 Enumeration Type Documentation

7.43.6.1 `enum mmc_switch_t`

MMC switch mode.

7.43.6.2 enum `sd_switch_t`

SDC switch mode.

7.43.6.3 enum `sd_switch_function_t`

SDC switch function.

7.43.6.4 enum `sdcbusmode_t`

Type of SDIO bus mode.

7.43.6.5 enum `sdcbusclk_t`

Max supported clock.

7.43.7 Function Documentation

7.43.7.1 static bool `mode_detect (SDCDriver * sdc)` [static]

Detects card mode.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	---

Returns

The operation status.

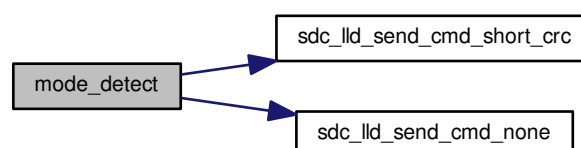
Return values

<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:



7.43.7.2 static bool mmc_init (**SDCDriver** * *sdcp*) [static]

Init procedure for MMC.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver 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:



7.43.7.3 static bool sdc_init (**SDCDriver** * *sdcp*) [static]

Init procedure for SDC.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Returns

The operation status.

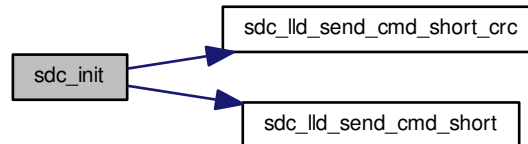
Return values

<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:



7.43.7.4 `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.

7.43.7.5 `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.

7.43.7.6 `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.

7.43.7.7 `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.

7.43.7.8 `static bool sdc_detect_bus_clk (SDCDriver* sdc, sdcbusclk_t* clk) [static]`

Reads supported bus clock and switch SDC to appropriate mode.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
out	<i>clk</i>	pointer to clock enum

Returns

The operation status.

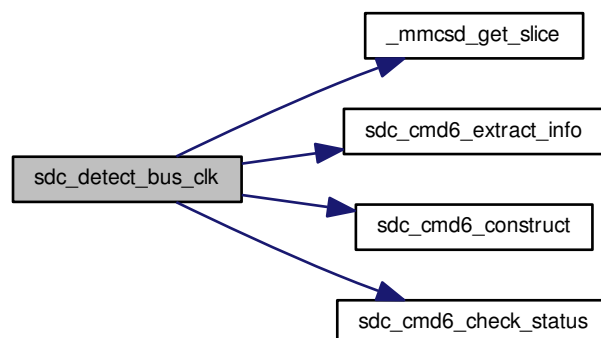
Return values

<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:



7.43.7.9 `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 SDCDriver object
out	<i>clk</i>	pointer to clock enum

Returns

The operation status.

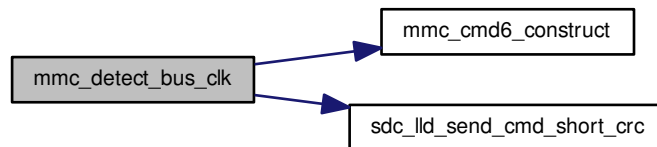
Return values

<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:



7.43.7.10 `static bool detect_bus_clk (SDCDriver * sdc, sdcbusclk_t * clk)` [static]

Reads supported bus clock and switch card to appropriate mode.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
out	<i>clk</i>	pointer to clock enum

Returns

The operation status.

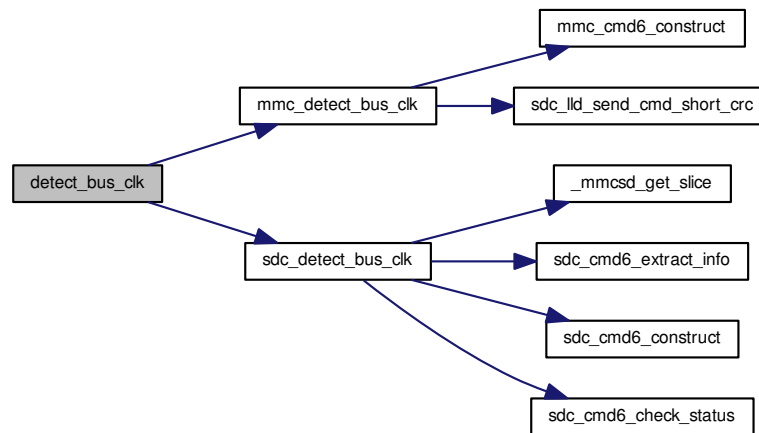
Return values

<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:



7.43.7.11 static bool sdc_set_bus_width (SDCDriver * *sdc*) [static]

Sets bus width for SDC.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	---

Returns

The operation status.

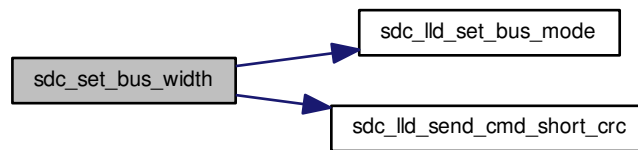
Return values

<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:



7.43.7.12 `static bool mmc_set_bus_width (SDCDriver * sdcp) [static]`

Sets bus width for MMC.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Returns

The operation status.

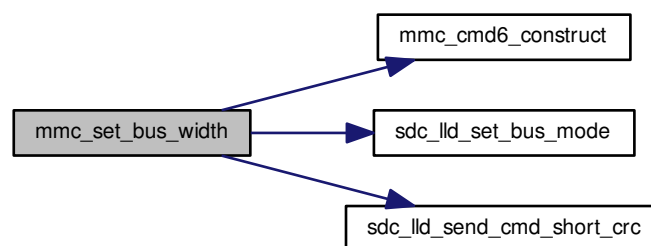
Return values

<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:



7.43.7.13 `bool _sdc_wait_for_transfer_state (SDCDriver * sdc)`

Wait for the card to complete pending operations.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	---

Returns

The operation status.

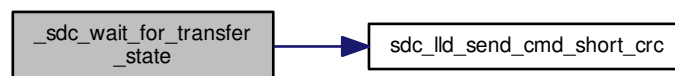
Return values

<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:



7.43.7.14 `void sdclnit (void)`

SDC Driver initialization.

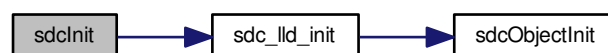
Note

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:



7.43.7.15 void sdcObjectInit (SDCDriver * *sdcp*)

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

Parameters

out	<i>sdcp</i>	pointer to the SDCDriver object
-----	-------------	---

Function Class:

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

7.43.7.16 void sdcStart (SDCDriver * *sdcp*, const SDCCConfig * *config*)

Configures and activates the SDC peripheral.

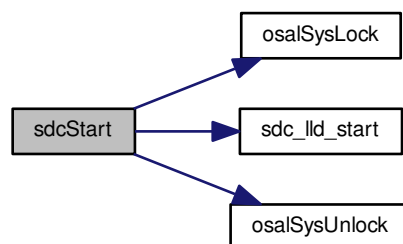
Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>config</i>	pointer to the SDCCConfig 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:

**7.43.7.17 void sdcStop (SDCDriver * *sdcp*)**

Deactivates the SDC peripheral.

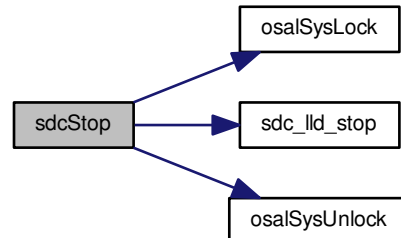
Parameters

in	<i>sdcp</i>	pointer to the SDCDriver 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:



7.43.7.18 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 SDCDriver object
----	------------	---

Returns

The operation status.

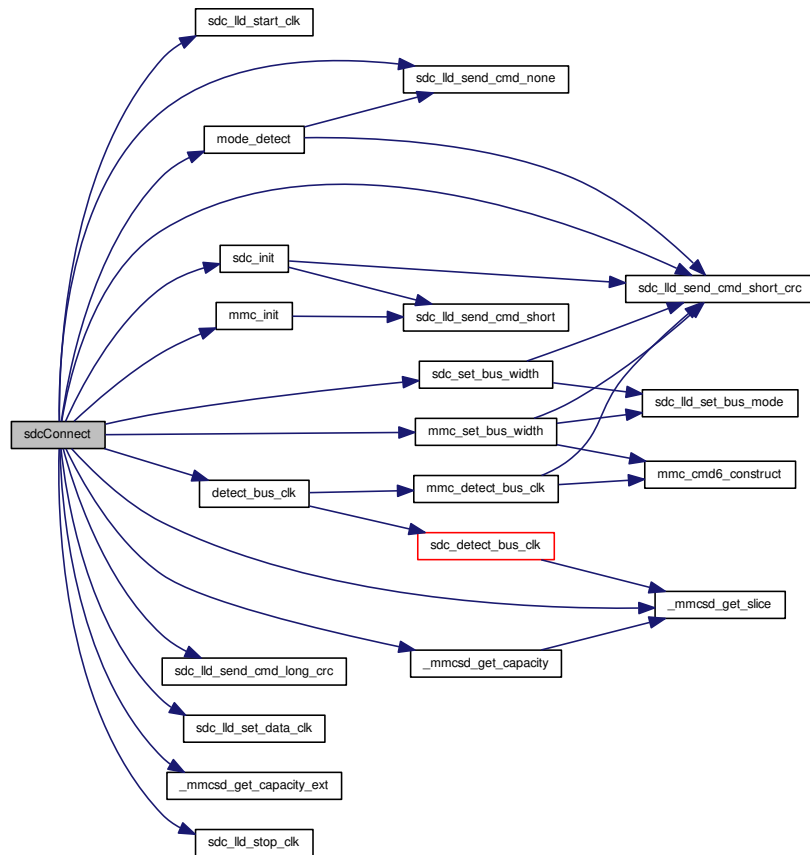
Return values

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	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:



7.43.7.19 bool sdcDisconnect (SDCDriver * sdc)

Brings the driver in a state safe for card removal.

Parameters

in	sdc	pointer to the SDC object
----	-----	---

Returns

The operation status.

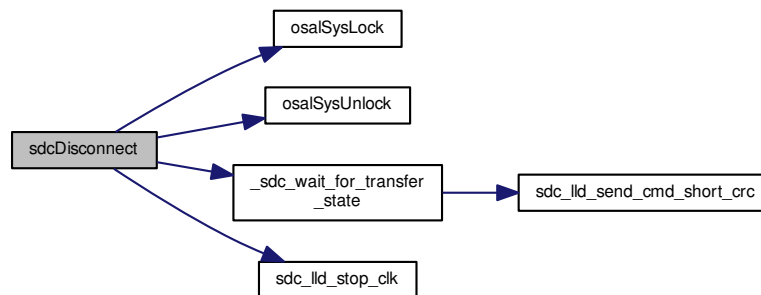
Return values

<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:



7.43.7.20 `bool sdcRead (SDCDriver * sdc, 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>sdc</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:

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:



7.43.7.21 `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

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	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:



7.43.7.22 `sdcflags_t sdcGetAndClearErrors (SDCDriver * sdc)`

Returns the errors mask associated to the previous operation.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	---

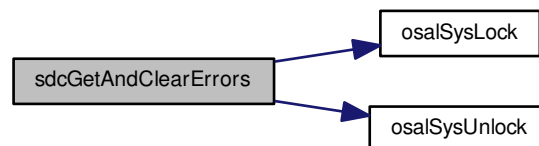
Returns

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:



7.43.7.23 `bool sdcSync (SDCDriver * sdc)`

Waits for card idle condition.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver 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:



7.43.7.24 bool sdcGetInfo (**SDCDriver** * *sdcp*, **BlockDeviceInfo** * *bdip*)

Returns the media info.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
out	<i>bdip</i>	pointer to a BlockDeviceInfo structure

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.

7.43.7.25 bool sdcErase (**SDCDriver** * *sdcp*, uint32_t *startblk*, uint32_t *endblk*)

Erases the supplied blocks.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver 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.
--------------------	--------------------------

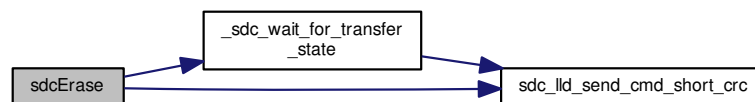
Return values

<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:



7.43.7.26 void sdc_ll_init (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:



7.43.7.27 void sdc_ll_start (SDCDriver * sdcp)

Configures and activates the SDC peripheral.

Parameters

in	<code>sdcp</code>	pointer to the <code>SDCDriver</code> object
----	-------------------	--

Function Class:

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

7.43.7.28 void sdc_lld_stop (SDCDriver * *sdcp*)

Deactivates the SDC peripheral.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Function Class:

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

7.43.7.29 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 SDCDriver object
----	-------------	---

Function Class:

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

7.43.7.30 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 SDCDriver object
in	<i>clk</i>	the clock mode

Function Class:

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

7.43.7.31 void sdc_lld_stop_clk (SDCDriver * *sdcp*)

Stops the SDIO clock.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Function Class:

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

7.43.7.32 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 SDCDriver object
in	<i>mode</i>	bus mode

Function Class:

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

7.43.7.33 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 SDCDriver 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.

7.43.7.34 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 SDCDriver 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

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.43.7.35 `bool sdc_ild_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 SDCDriver 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

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.43.7.36 `bool sdc_ild_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 SDCDriver 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

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.43.7.37 `bool sdc_lld_read (SDCDriver * sdcp, uint32_t startblk, uint8_t * buf, uint32_t n)`

Reads one or more blocks.

Parameters

in	<i>sdc</i> p	pointer to the SDCDriver 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:

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

7.43.7.38 `bool sdc_lld_write (SDCDriver * sdcp, uint32_t startblk, const uint8_t * buf, uint32_t n)`

Writes one or more blocks.

Parameters

in	<i>sdc</i> p	pointer to the SDCDriver 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.

7.43.7.39 `bool sdc_lld_sync (SDCDriver * sdcp)`

Waits for card idle condition.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver 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.

7.43.8 Variable Documentation

7.43.8.1 `const struct SDCDriverVMT sdc_vmt [static]`

Initial value:

```
= {
    (size_t)0,
    (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.

7.43.8.2 SDCDriver SDCD1

SDCD1 driver identifier.

7.44 Generic Sensor Interface

HAL Generic Sensor Interface.

7.44.1 Detailed Description

HAL Generic Sensor Interface.

Macros

- #define `_base_sensor_methods_alone`
BaseSensor specific methods.
- #define `_base_sensor_methods`
BaseSensor specific methods with inherited ones.
- #define `_base_sensor_data`
BaseSensor specific data.

Macro Functions (BaseSensor)

- #define `sensorGetChannelNumber(ip)` (ip)->vmt->get_channels_number(ip)
Sensors get channels number.
- #define `sensorReadRaw(ip, dp)` (ip)->vmt->read_raw(ip, dp)
Sensors read raw data.
- #define `sensorReadCooked(ip, dp)` (ip)->vmt->read_cooked(ip, dp)
Sensors read cooked data.

Data Structures

- struct `BaseSensorVMT`
BaseSensor virtual methods table.
- struct `BaseSensor`
Base stream class.

7.44.2 Macro Definition Documentation

7.44.2.1 #define _base_sensor_methods_alone

Value:

```
/* Get number of channels.*/
size_t (*get_channels_number)(void *instance);
/* Reads the sensor raw data.*/
msg_t (*read_raw)(void *instance, int32_t axes[]);
/* Reads the sensor returning normalized data.*/
msg_t (*read_cooked)(void *instance, float axes[]);
```

`BaseSensor` specific methods.

7.44.2.2 #define _base_sensor_methods

Value:

```
_base_object_methods \
_base_sensor_methods_alone
```

[BaseSensor](#) specific methods with inherited ones.

7.44.2.3 #define _base_sensor_data

[BaseSensor](#) specific data.

Note

It is empty because [BaseSensor](#) is only an interface without implementation.

7.44.2.4 #define sensorGetChannelNumber(ip) (ip)->vmt->get_channels_number(ip)

Sensors get channels number.

Parameters

in	<i>ip</i>	pointer to a BaseSensor or derived class.
----	-----------	---

Returns

The number of channels of the [BaseSensor](#)

Function Class:

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

7.44.2.5 #define sensorReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)

Sensors read raw data.

Parameters

in	<i>ip</i>	pointer to a BaseSensor or derived class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.44.2.6 `#define sensorReadCooked(ip, dp)(ip)->vmt->read_cooked(ip, dp)`

Sensors read cooked data.

Parameters

in	<i>ip</i>	pointer to a BaseSensor or derived class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.45 Serial Driver

Generic Serial Driver.

7.45.1 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.

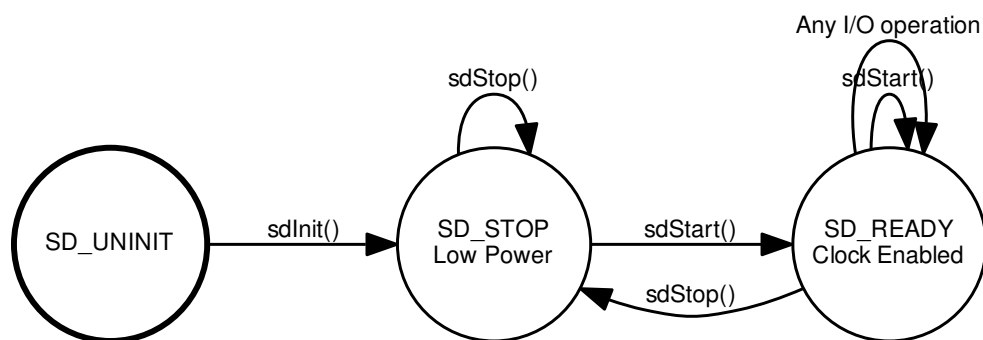
The module also contains functions that make the implementation of the interrupt service routines much easier.

Precondition

In order to use the SERIAL driver the `HAL_USE_SERIAL` option must be enabled in [halconf.h](#).

7.45.2 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`

- *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.*
- #define `SD_QUEUE_FULL_ERROR` (`eventflags_t`)1024
- *Queue full.*

Serial configuration options

- #define `SERIAL_DEFAULT_BITRATE` 38400
- *Default bit rate.*
- #define `SERIAL_BUFFERS_SIZE` 16
- *Serial buffers size.*

Macro Functions

- #define `sdPutl`(sdp, b) `oqPutl`(&(sdp)->oqueue, b)
- *Direct write to a [SerialDriver](#).*
- #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 `sdGetl`(sdp) `iqGetl`(&(sdp)->iqueue)
- *Direct read from a [SerialDriver](#).*
- #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 `sdWritel`(sdp, b, n) `oqWritel`(&(sdp)->oqueue, b, n)
- *Direct blocking write to a [SerialDriver](#).*
- #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 `sdReadl`(sdp, b, n) `iqReadl`(&(sdp)->iqueue, b, n, TIME_INFINITE)
- *Direct blocking read from 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 sdIncomingDataI (SerialDriver *sdp, uint8_t b)`
Handles incoming data.
- `msg_t sdRequestDataI (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.
- `msg_t sdControl (SerialDriver *sdp, unsigned int operation, void *arg)`
Control operation on a serial port.
- `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

Variables

- [SerialDriver SD1](#)
USART1 serial driver identifier.
- static const [SerialConfig default_config](#)
Driver default configuration.

7.45.3 Macro Definition Documentation

7.45.3.1 #define SD_PARITY_ERROR (eventflags_t)32

Parity.

7.45.3.2 #define SD_FRAMING_ERROR (eventflags_t)64

Framing.

7.45.3.3 #define SD_OVERRUN_ERROR (eventflags_t)128

Overflow.

7.45.3.4 #define SD_NOISE_ERROR (eventflags_t)256

Line noise.

7.45.3.5 #define SD_BREAK_DETECTED (eventflags_t)512

LIN Break.

7.45.3.6 #define SD_QUEUE_FULL_ERROR (eventflags_t)1024

Queue full.

7.45.3.7 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.45.3.8 #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.

This is a global setting and it can be overridden by low level driver specific settings.

7.45.3.9 `#define _serial_driver_methods_base_asynchronous_channel_methods`

`SerialDriver` specific methods.

7.45.3.10 `#define sdPutl(sdp, b) oqPutl(&(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.

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.

7.45.3.11 `#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.

Function Class:

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

7.45.3.12 `#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.

Function Class:

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

7.45.3.13 `#define sdGetl(sdp) iqGetl(&(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.

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.

7.45.3.14 #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.

Function Class:

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

7.45.3.15 #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.

Function Class:

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

7.45.3.16 #define sdWritel(*sdp*, *b*, *n*) oqWritel(&(sdp)->oqueue, b, n)

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.

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.

7.45.3.17 #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.

Function Class:

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

7.45.3.18 `#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.

Function Class:

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

7.45.3.19 `#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.

Function Class:

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

7.45.3.20 `#define sdRead(sdp, b, n) iqReadI(&(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.

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.

7.45.3.21 `#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.

Function Class:

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

7.45.3.22 `#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.

Function Class:

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

7.45.3.23 `#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.

Function Class:

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

7.45.3.24 `#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`.

7.45.3.25 `#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.

7.45.4 Typedef Documentation

7.45.4.1 `typedef struct SerialDriver SerialDriver`

Structure representing a serial driver.

7.45.5 Enumeration Type Documentation

7.45.5.1 enum sdstate_t

Driver state machine possible states.

Enumerator

SD_UNINIT Not initialized.

SD_STOP Stopped.

SD_READY Ready.

7.45.6 Function Documentation

7.45.6.1 void sdInit (void)

Serial Driver initialization.

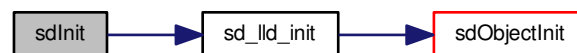
Note

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:



7.45.6.2 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.

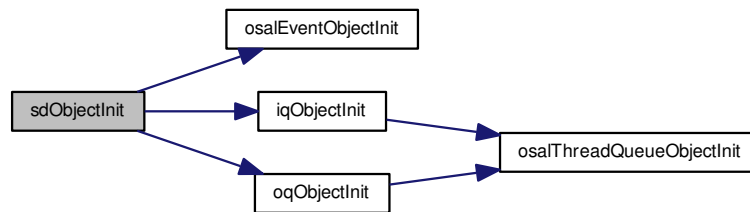
Parameters

out	<i>sdp</i>	pointer to a SerialDriver 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 <code>NULL</code> .
in	<i>onotify</i>	pointer to a callback function that is invoked when some data is written in the Queue. The value can be <code>NULL</code> .

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:



7.45.6.3 void sdStart (SerialDriver * sdp, const SerialConfig * config)

Configures and starts the driver.

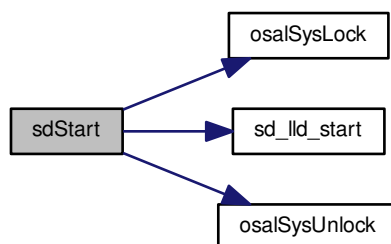
Parameters

in	<i>sdp</i>	pointer to a SerialDriver object
in	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to <code>NULL</code> 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:



7.45.6.4 void sdStop (SerialDriver * sdp)

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message `MSG_RESET`.

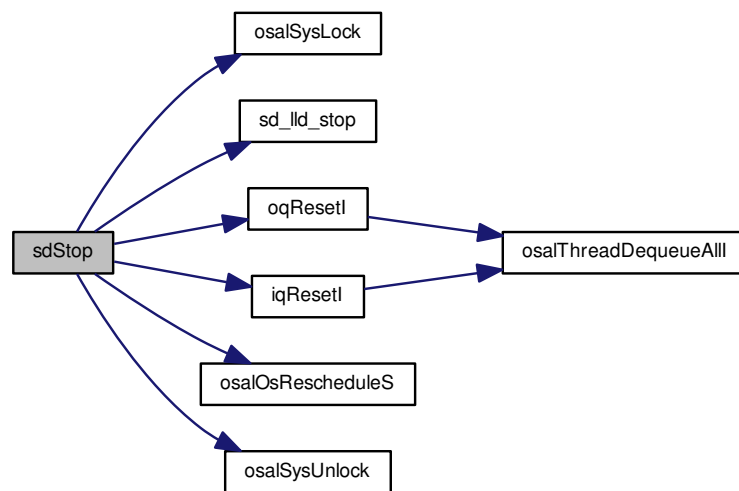
Parameters

in	<i>sdp</i>	pointer to a SerialDriver 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:

7.45.6.5 void sdIncomingData (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.

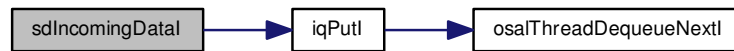
Parameters

in	<i>sdp</i>	pointer to a SerialDriver 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:



7.45.6.6 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 SerialDriver structure
----	------------	---

Returns

The byte value read from the driver's output queue.

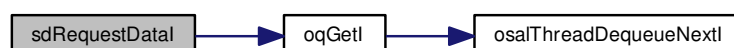
Return values

<code>MSG_TIMEOUT</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:



7.45.6.7 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	<i>sdp</i>	pointer to a SerialDriver structure
----	------------	---

Returns

The queue status.

Return values

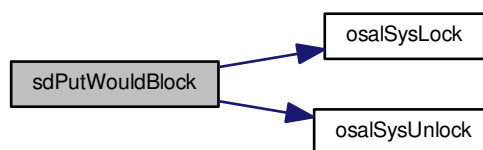
<i>false</i>	if the next write operation would not block.
<i>true</i>	if the next write operation would block.

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:



7.45.6.8 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 SerialDriver structure
----	------------	---

Returns

The queue status.

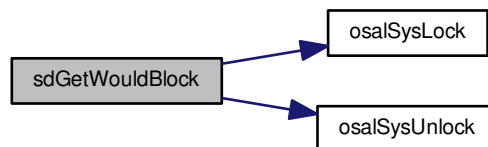
Return values

<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:

**7.45.6.9 msg_t sdControl (SerialDriver * sdp, unsigned int operation, void * arg)**

Control operation on a serial port.

Parameters

in	<i>sdp</i>	pointer to a SerialDriver object
in	<i>operation</i>	control operation code
in, out	<i>arg</i>	operation argument

Returns

The control operation status.

Return values

<i>MSG_OK</i>	in case of success.
---------------	---------------------

Return values

<i>MSG_TIMEOUT</i>	in case of operation timeout.
<i>MSG_RESET</i>	in case of operation reset.

Function Class:

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

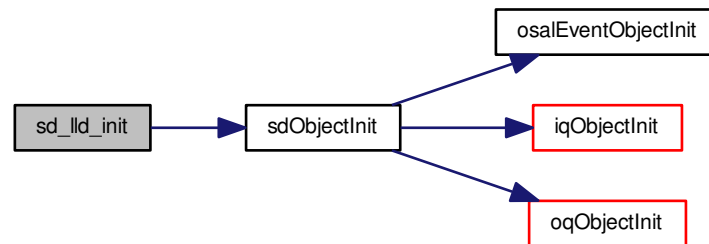
7.45.6.10 void sd_lld_init (void)

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:



7.45.6.11 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 SerialDriver object
in	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to <code>NULL</code> then a default configuration is used.

Function Class:

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

7.45.6.12 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 SerialDriver object
----	------------	--

Function Class:

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

7.45.7 Variable Documentation

7.45.7.1 SerialDriver SD1

USART1 serial driver identifier.

7.45.7.2 `const SerialConfig default_config` `[static]`

Initial value:

```
= {  
    38400  
}
```

Driver default configuration.

7.46 Serial over USB Driver

Serial over USB Driver.

7.46.1 Detailed Description

Serial over USB Driver.

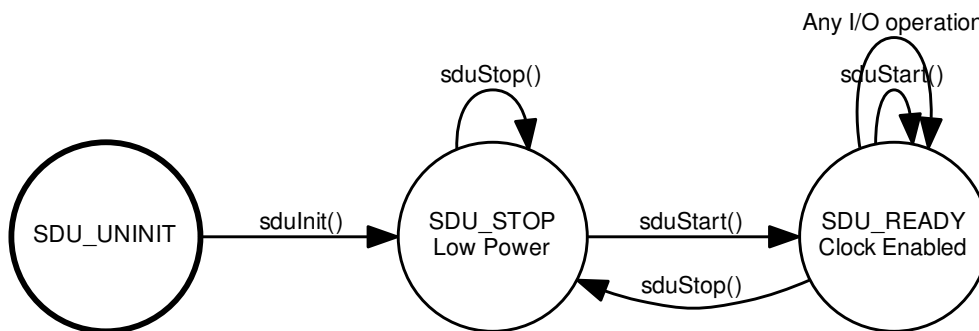
This module implements an USB Communication Device Class (CDC) as a normal serial communication port accessible from the device application.

Precondition

In order to use the USB over Serial driver the `HAL_USE_SERIAL_USB` option must be enabled in `halconf.h`.

7.46.2 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 [sduSuspendHookI](#) ([SerialUSBDriver](#) *sdup)
USB device suspend handler.
- void [sduWakeupHookI](#) ([SerialUSBDriver](#) *sdup)
USB device wakeup 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.
- [msg_t](#) [sduControl](#) ([USBDriver](#) *usbp, unsigned int operation, void *arg)
Control operation on a serial USB port.

Enumerations

7.46.3 Macro Definition Documentation

7.46.3.1 #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.

7.46.3.2 #define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.46.3.3 #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 SerialUSBConfig *config;
```

`SerialDriver` specific data.

7.46.3.4 #define _serial_usb_driver_methods _base_asynchronous_channel_methods

`SerialUSBDriver` specific methods.

7.46.4 Typedef Documentation

7.46.4.1 typedef struct SerialUSBDriver SerialUSBDriver

Structure representing a serial over USB driver.

7.46.5 Enumeration Type Documentation

7.46.5.1 enum sdustate_t

Driver state machine possible states.

Enumerator

SDU_UNINIT Not initialized.

SDU_STOP Stopped.

SDU_READY Ready.

7.46.6 Function Documentation

7.46.6.1 static void ibnotify (io_buffers_queue_t * bqp) [static]

Notification of empty buffer released into the input buffers queue.

Parameters

in	<i>bqp</i>	the buffers queue pointer.
----	------------	----------------------------

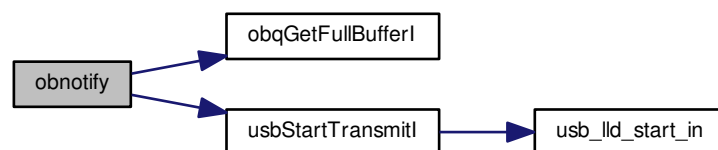
7.46.6.2 static void obnotify (io_buffers_queue_t * bqp) [static]

Notification of filled buffer inserted into the output buffers queue.

Parameters

in	<i>bqp</i>	the buffers queue pointer.
----	------------	----------------------------

Here is the call graph for this function:



7.46.6.3 void sdulnit (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.

7.46.6.4 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.

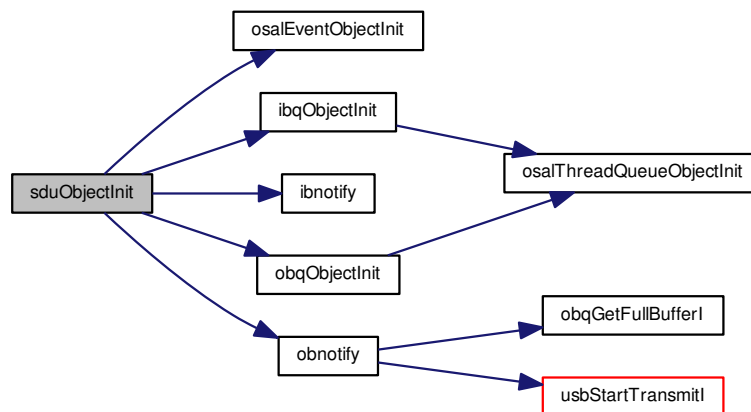
Parameters

out	<i>sdup</i>	pointer to a <code>SerialUSBDriver</code> 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:

**7.46.6.5 void sduStart (SerialUSBDriver * sdup, const SerialUSBConfig * config)**

Configures and starts the driver.

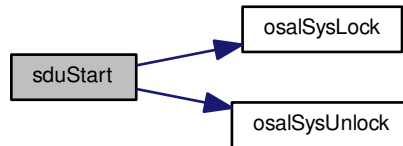
Parameters

in	<i>sdup</i>	pointer to a <code>SerialUSBDriver</code> 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:



7.46.6.6 void sduStop (SerialUSBDriver * sdup)

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message `MSG_RESET`.

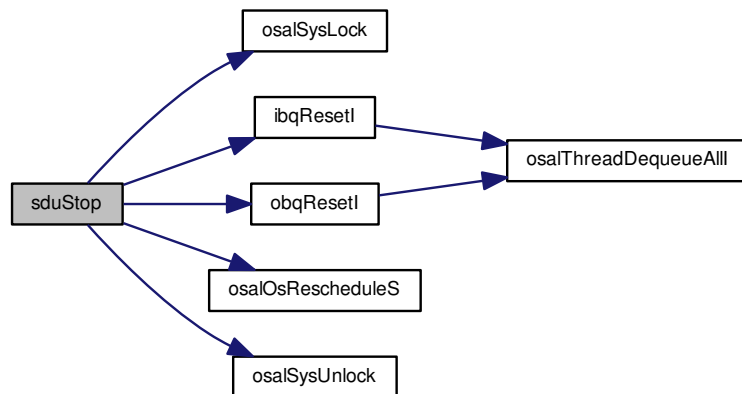
Parameters

in	<i>sdup</i>	pointer to a SerialUSBDriver 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:



7.46.6.7 void sduSuspendHookI (SerialUSBDriver * sdup)

USB device suspend handler.

Generates a `CHN_DISCONNECT` event and puts queues in non-blocking mode, this way the application cannot get stuck in the middle of an I/O operations.

Note

If this function is not called from an ISR then an explicit call to `osalOsRescheduleS()` is necessary afterward.

Parameters

in	<i>sdup</i>	pointer to a <code>SerialUSBDriver</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.

7.46.6.8 void sduWakeupHookI (SerialUSBDriver * sdup)

USB device wakeup handler.

Generates a `CHN_CONNECT` event and resumes normal queues operations.

Note

If this function is not called from an ISR then an explicit call to `osalOsRescheduleS()` is necessary afterward.

Parameters

in	<i>sdup</i>	pointer to a <code>SerialUSBDriver</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.

7.46.6.9 void sduConfigureHookI (SerialUSBDriver * sdup)

USB device configured handler.

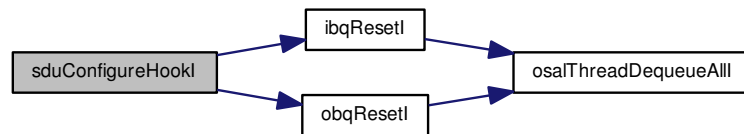
Parameters

in	<i>sdup</i>	pointer to a <code>SerialUSBDriver</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:



7.46.6.10 bool sduRequestsHook (USBDriver * *usbp*)

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 USBDriver object
----	-------------	---

Returns

The hook status.

Return values

<i>true</i>	Message handled internally.
<i>false</i>	Message not handled.

7.46.6.11 void sduSOFHookI (SerialUSBDriver * *sdup*)

SOF handler.

The SOF interrupt is used for automatic flushing of incomplete buffers pending in the output queue.

Parameters

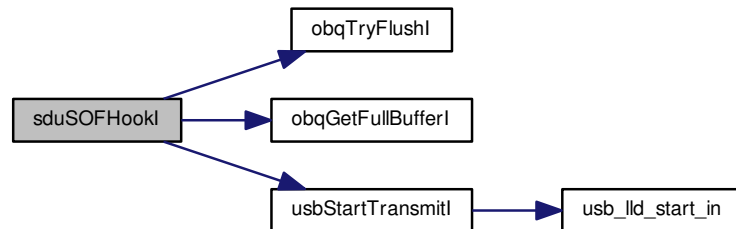
in	<i>sdup</i>	pointer to a SerialUSBDriver object
----	-------------	---

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:



7.46.6.12 void sduDataTransmitted (USBDriver * usbp, usbep_t ep)

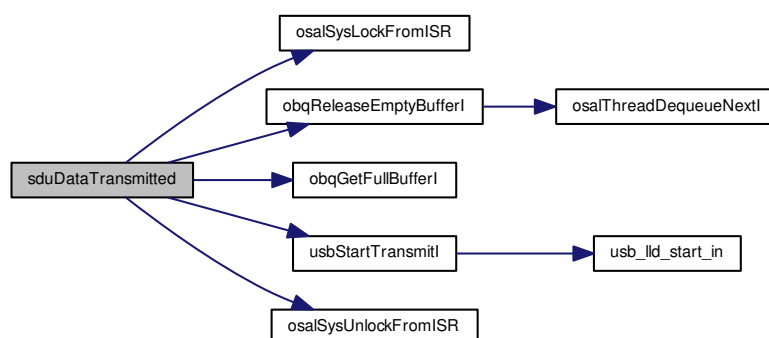
Default data transmitted callback.

The application must use this function as callback for the IN data endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	IN endpoint number

Here is the call graph for this function:



7.46.6.13 void sduDataReceived (USBDriver * usbp, usbep_t ep)

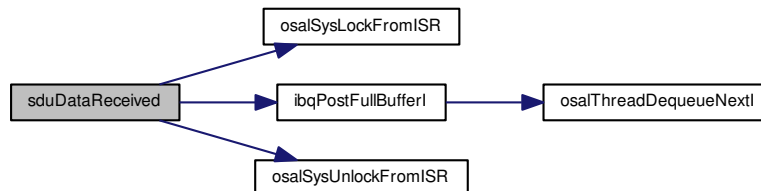
Default data received callback.

The application must use this function as callback for the OUT data endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	OUT endpoint number

Here is the call graph for this function:



7.46.6.14 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 USBDriver object
in	<i>ep</i>	endpoint number

7.46.6.15 [msg_t](#) sduControl ([USBDriver](#) * *usbp*, unsigned int *operation*, void * *arg*)

Control operation on a serial USB port.

Parameters

in	<i>usbp</i>	pointer to a USBDriver object
in	<i>operation</i>	control operation code
in, out	<i>arg</i>	operation argument

Returns

The control operation status.

Return values

<i>MSG_OK</i>	in case of success.
<i>MSG_TIMEOUT</i>	in case of operation timeout.
<i>MSG_RESET</i>	in case of operation reset.

Function Class:

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

7.47 SPI Driver

Generic SPI Driver.

7.47.1 Detailed Description

Generic SPI Driver.

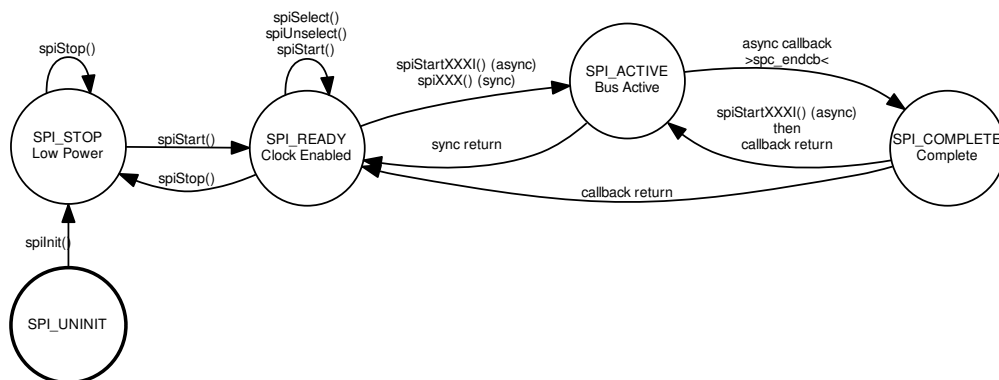
This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

Precondition

In order to use the SPI driver the `HAL_USE_SPI` option must be enabled in `halconf.h`.

7.47.2 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.

Macros

- `#define SPI_SUPPORTS_CIRCULAR TRUE`
Circular mode support flag.

Chip Select modes

- `#define SPI_SELECT_MODE_NONE`
- `#define SPI_SELECT_MODE_PAD 1` /** @brief Legacy mode. */
- `#define SPI_SELECT_MODE_PORT 2` /** @brief Fastest mode. */

- #define **SPI_SELECT_MODE_LINE** 3 /** @brief Packed mode. */
- #define **SPI_SELECT_MODE_LLD** 4 /** @brief LLD-defined mode.*/

SPI configuration options

- #define **SPI_USE_WAIT** TRUE
Enables synchronous APIs.
- #define **SPI_USE_CIRCULAR** FALSE
Enables circular transfers APIs.
- #define **SPI_USE_MUTUAL_EXCLUSION** TRUE
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.
- #define **SPI_SELECT_MODE** SPI_SELECT_MODE_PAD
Handling method for SPI CS line.

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 **spiStartReceiveI**(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.
- #define **_spi_isr_code_half1**(spip)
Common ISR code in circular mode.
- #define **_spi_isr_code_half2**(spip)
Common ISR code in circular mode.

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 [SPIDriver](#)
Driver configuration structure.
- struct [SPIConfig](#)
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 [spiAbortI](#) ([SPIDriver](#) *spip)
Aborts the ongoing SPI operation.
- void [spiAbort](#) ([SPIDriver](#) *spip)
Aborts the ongoing SPI operation, if any.
- void [spiIgnore](#) ([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.
- void `spi_lld_abort` (`SPIDriver *spip`)
Aborts the ongoing SPI operation, if any.
- `uint16_t spi_lld_polled_exchange` (`SPIDriver *spip`, `uint16_t frame`)
Exchanges one frame using a polled wait.

Enumerations

Variables

- `SPIDriver SPI1`
SPI1 driver identifier.

7.47.3 Macro Definition Documentation

7.47.3.1 `#define SPI_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.47.3.2 `#define SPI_USE_CIRCULAR FALSE`

Enables circular transfers APIs.

Note

Disabling this option saves both code and data space.

7.47.3.3 #define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.47.3.4 #define SPI_SELECT_MODE SPI_SELECT_MODE_PAD

Handling method for SPI CS line.

Note

Disabling this option saves both code and data space.

7.47.3.5 #define spiSelectl(*spip*)

Value:

```
do {
    spi_llc_select(spip);
} while (false)
```

\\

Asserts the slave select signal and prepares for transfers.

Parameters

in	<i>spip</i>	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.

7.47.3.6 #define spiUnselectl(*spip*)

Value:

```
do {
    spi_llc_unselect(spip);
} while (false)
```

\\

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in	<i>spip</i>	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.

7.47.3.7 #define spiStartIgnore(*spip*, *n*)**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_llc_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 <code>SPIDriver</code> 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.

7.47.3.8 #define spiStartExchange(*spip*, *n*, *txbuf*, *rxbuf*)**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_llc_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 SPIDriver 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.

7.47.3.9 #define spiStartSendl(*spip*, *n*, *txbuf*)**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_llc_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 SPIDriver 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.

7.47.3.10 #define spiStartReceiveI(*spip*, *n*, *rxbuf*)**Value:**

```

{
    (spip)->state = SPI_ACTIVE;
    spi_llc_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.

7.47.3.11 `#define spiPolledExchange(spip, frame) spi_llc_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.

7.47.3.12 #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 SPIDriver object
----	-------------	---

Function Class:

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

7.47.3.13 #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 SPIDriver object
----	-------------	---

Function Class:

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

7.47.3.14 `#define _spi_isr_code_half1(spip)`**Value:**

```

{
    if ((spip)->config->end_cb) {
        (spip)->config->end_cb(spip);
    }
}

```

Common ISR code in circular mode.

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>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.47.3.15 `#define _spi_isr_code_half2(spip)`**Value:**

```

{
    if ((spip)->config->end_cb) {
        (spip)->state = SPI_COMPLETE;
        (spip)->config->end_cb(spip);
        if ((spip)->state == SPI_COMPLETE)
            (spip)->state = SPI_ACTIVE;
    }
}

```

Common ISR code in circular mode.

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>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.47.3.16 #define SPI_SUPPORTS_CIRCULAR TRUE

Circular mode support flag.

7.47.3.17 #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`.

7.47.4 Typedef Documentation**7.47.4.1 typedef struct SPIDriver SPIDriver**

Type of a structure representing an SPI driver.

7.47.4.2 typedef void(* spicallback_t) (SPIDriver *spip)

SPI notification callback type.

Parameters

in	spip	pointer to the SPIDriver object triggering the callback
----	------	---

7.47.5 Enumeration Type Documentation**7.47.5.1 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.

7.47.6 Function Documentation**7.47.6.1 void spilnit (void)**

SPI Driver initialization.

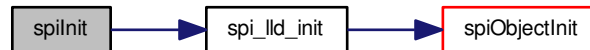
Note

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:

**7.47.6.2 void spiObjectInit (SPIDriver * *spip*)**

Initializes the standard part of a `SPIDriver` structure.

Parameters

out	<i>spip</i>	pointer to the <code>SPIDriver</code> 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:

**7.47.6.3 void spiStart (SPIDriver * *spip*, const SPIConfig * *config*)**

Configures and activates the SPI peripheral.

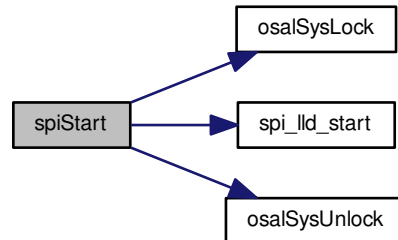
Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>config</i>	pointer to the <code>SPIConfig</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:



7.47.6.4 void spiStop (SPIDriver * *spip*)

Deactivates the SPI peripheral.

Note

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

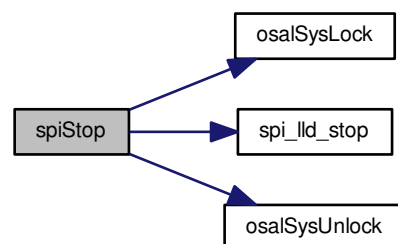
Parameters

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:



7.47.6.5 void spiSelect (SPIDriver * *spip*)

Asserts the slave select signal and prepares for transfers.

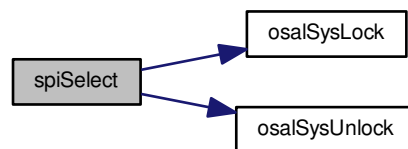
Parameters

in	<i>spip</i>	pointer to the SPIDriver 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:

**7.47.6.6 void spiUnselect (SPIDriver * *spip*)**

Deasserts the slave select signal.

The previously selected peripheral is unselected.

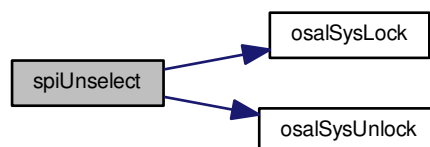
Parameters

in	<i>spip</i>	pointer to the SPIDriver 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:



7.47.6.7 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.

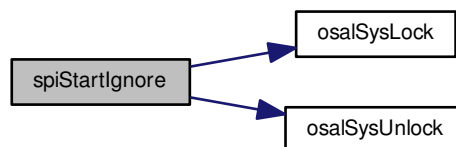
Parameters

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:



7.47.6.8 void spiStartExchange (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.

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.

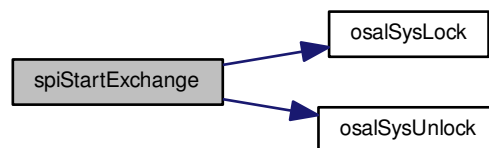
Parameters

in	<i>spip</i>	pointer to the SPIDriver 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:

**7.47.6.9 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.

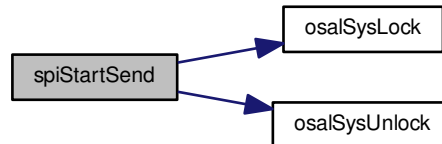
Parameters

in	<i>spip</i>	pointer to the SPIDriver 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:



7.47.6.10 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.

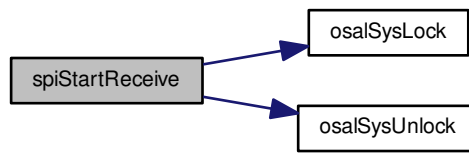
Parameters

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:



7.47.6.11 void spiAbortl (SPIDriver * *spip*)

Aborts the ongoing SPI operation.

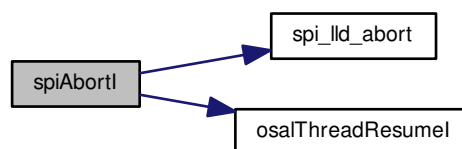
Parameters

in	<i>spip</i>	pointer to the SPIDriver 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:



7.47.6.12 void spiAbort (SPIDriver * *spip*)

Aborts the ongoing SPI operation, if any.

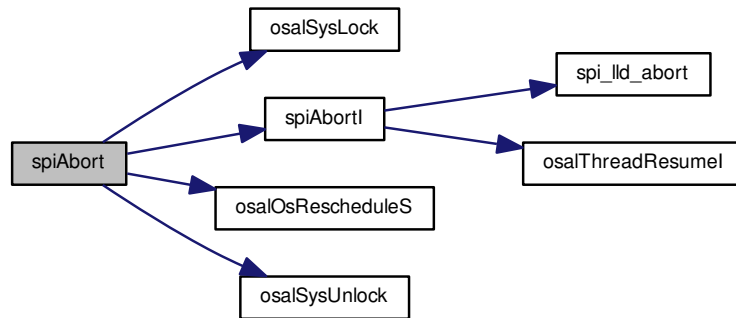
Parameters

in	<i>spip</i>	pointer to the SPIDriver 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:



7.47.6.13 void spignore (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`).

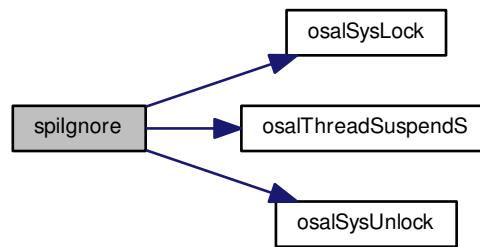
Parameters

in	<i>spip</i>	pointer to the SPIDriver 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:



7.47.6.14 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.

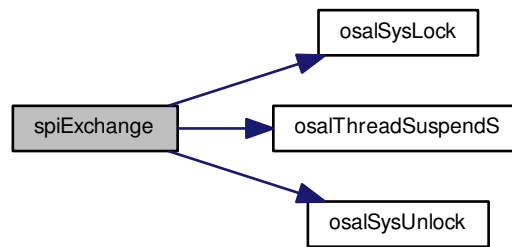
Parameters

in	<i>spip</i>	pointer to the SPIDriver 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:



7.47.6.15 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.

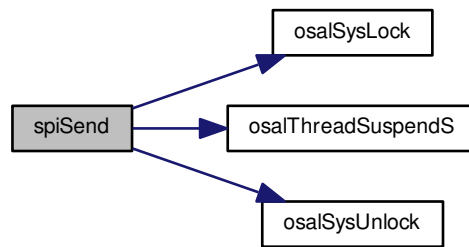
Parameters

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:



7.47.6.16 void spiReceive (SPIDriver * *spip*, size_t *n*, void * *rxbuf*)

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.

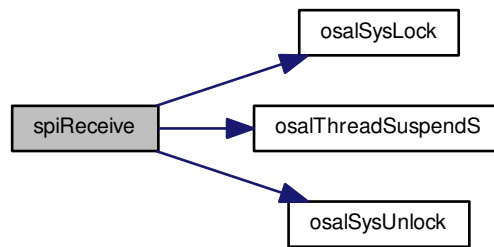
Parameters

in	<i>spip</i>	pointer to the SPIDriver 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:



7.47.6.17 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 SPIDriver 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:



7.47.6.18 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 SPIDriver 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:

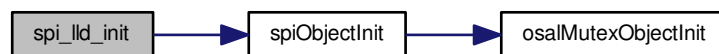
**7.47.6.19 void spi_lld_init (void)**

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:

**7.47.6.20 void spi_lld_start (SPIDriver * spip)**

Configures and activates the SPI peripheral.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.47.6.21 void spi_lld_stop (SPIDriver * *spip*)

Deactivates the SPI peripheral.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.47.6.22 void spi_lld_select (SPIDriver * *spip*)

Asserts the slave select signal and prepares for transfers.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.47.6.23 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 SPIDriver object
----	-------------	---

Function Class:

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

7.47.6.24 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 SPIDriver object
in	<i>n</i>	number of words to be ignored

Function Class:

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

7.47.6.25 void `spi_ll_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 SPIDriver 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.

7.47.6.26 void `spi_ll_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 SPIDriver 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.

7.47.6.27 void spi_llc_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 SPIDriver object
in	<i>n</i>	number of words to receive
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

7.47.6.28 void spi_llc_abort (SPIDriver * *spip*)

Aborts the ongoing SPI operation, if any.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---------------------------------

Function Class:

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

7.47.6.29 uint16_t spi_llc_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 SPIDriver object
in	<i>frame</i>	the data frame to send over the SPI bus

Returns

The received data frame from the SPI bus.

Function Class:

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

7.47.7 Variable Documentation**7.47.7.1 SPIDriver SPID1**

SPI1 driver identifier.

7.48 ST Driver

Generic System Tick Driver.

7.48.1 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 (sys_time_t abstime)`
Starts the alarm.
- `void stStopAlarm (void)`
Stops the alarm interrupt.
- `void stSetAlarm (sys_time_t abstime)`
Sets the alarm time.
- `sys_time_t stGetAlarm (void)`
Returns the current alarm time.
- `void st_lld_init (void)`
Low level ST driver initialization.
- `static sys_time_t st_lld_get_counter (void)`
Returns the time counter value.
- `static void st_lld_start_alarm (sys_time_t abstime)`
Starts the alarm.
- `static void st_lld_stop_alarm (void)`
Stops the alarm interrupt.
- `static void st_lld_set_alarm (sys_time_t abstime)`
Sets the alarm time.
- `static sys_time_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.

7.48.2 Macro Definition Documentation

7.48.2.1 `#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.

7.48.2.2 #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.

7.48.3 Function Documentation**7.48.3.1 void stInit (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:



7.48.3.2 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:



7.48.3.3 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:



7.48.3.4 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:



7.48.3.5 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:



7.48.3.6 void st_llid_init (void)

Low level ST driver initialization.

Function Class:

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

7.48.3.7 static systime_t st_llid_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.

7.48.3.8 static void st_llid_start_alarm (systime_t abstime) [inline],[static]

Starts the alarm.

Note

Makes sure that no spurious alarms are triggered after this call.

Parameters

in	<i>abstime</i>	the time to be set for the first alarm
----	----------------	--

Function Class:

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

7.48.3.9 static void st_llid_stop_alarm (void) [inline],[static]

Stops the alarm interrupt.

Function Class:

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

7.48.3.10 static void st_llid_set_alarm (systime_t abstime) [inline],[static]

Sets the alarm time.

Parameters

in	<i>abstime</i>	the time to be set for the next alarm
----	----------------	---------------------------------------

Function Class:

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

7.48.3.11 `static systime_t st_ild_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.

7.48.3.12 `static bool st_ild_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.

7.49 Generic Thermometer Interface

HAL Generic Thermometer Interface.

7.49.1 Detailed Description

HAL Generic Thermometer Interface.

Macros

- `#define _base_thermometer_methods_alone`
BaseThermometer specific methods.
- `#define _base_thermometer_methods`
BaseThermometer specific methods with inherited ones.
- `#define _base_thermometer_data_base_sensor_data`
BaseThermometer specific data.

Macro Functions (BaseThermometer)

- `#define thermometerGetChannelsNumber(ip) (ip)->vmt->get_channels_number(ip)`
Thermometer get channels number.
- `#define thermometerReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)`
Thermometer read raw data.
- `#define thermometerReadCooked(ip, dp) (ip)->vmt->read_cooked(ip, dp)`
Thermometer read cooked data.
- `#define thermometerSetBias(ip, bp) (ip)->vmt->set_bias(ip, bp)`
Updates thermometer bias data from received buffer.
- `#define thermometerResetBias(ip) (ip)->vmt->reset_bias(ip)`
Reset thermometer bias data restoring it to zero.
- `#define thermometerSetSensitivity(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)`
Updates thermometer sensitivity data from received buffer.
- `#define thermometerResetSensitivity(ip) (ip)->vmt->reset_sensitivity(ip)`
Reset thermometer sensitivity data restoring it to its typical value.

Data Structures

- struct `BaseThermometerVMT`
BaseThermometer virtual methods table.
- struct `BaseThermometer`
Base thermometer class.

7.49.2 Macro Definition Documentation

7.49.2.1 `#define _base_thermometer_methods_alone`

Value:


```

/* Invoke the set bias procedure.*/
msg_t (*set_bias)(void *instance, float biases[]);
/* Remove bias stored data.*/
msg_t (*reset_bias)(void *instance);
/* Invoke the set sensitivity procedure.*/
msg_t (*set_sensitivity)(void *instance, float sensitivities[]);
/* Restore sensitivity stored data to default.*/
msg_t (*reset_sensitivity)(void *instance);

```

[BaseThermometer](#) specific methods.

7.49.2.2 #define _base_thermometer_methods

Value:

```

_base_sensor_methods
_base_thermometer_methods_alone

```

[BaseThermometer](#) specific methods with inherited ones.

7.49.2.3 #define _base_thermometer_data _base_sensor_data

[BaseThermometer](#) specific data.

7.49.2.4 #define thermometerGetChannelsNumber(ip) (ip)->vmt->get_channels_number(ip)

Thermometer get channels number.

Parameters

in	<i>ip</i>	pointer to a BaseThermometer class.
----	-----------	---

Returns

The number of channels of the [BaseThermometer](#)

Function Class:

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

7.49.2.5 #define thermometerReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)

Thermometer read raw data.

Parameters

in	<i>ip</i>	pointer to a BaseThermometer class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.49.2.6 #define thermometerReadCooked(ip, dp) (ip)->vmt->read_cooked(ip, dp)

Thermometer read cooked data.

Parameters

in	<i>ip</i>	pointer to a BaseThermometer class.
in	<i>dp</i>	pointer to a data array.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.49.2.7 #define thermometerSetBias(ip, bp) (ip)->vmt->set_bias(ip, bp)

Updates thermometer bias data from received buffer.

Note

The bias buffer must have the same length of the the thermometer channels number.

Parameters

in	<i>ip</i>	pointer to a BaseThermometer class.
in	<i>bp</i>	pointer to a buffer of bias values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.49.2.8 #define thermometerResetBias(*ip*)(ip)->vmt->reset_bias(ip)

Reset thermometer bias data restoring it to zero.

Parameters

in	<i>ip</i>	pointer to a BaseThermometer class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.49.2.9 #define thermometerSetSensitivity(*ip, sp*)(ip)->vmt->set_sensitivity(ip, sp)

Updates thermometer sensitivity data from received buffer.

Note

The sensitivity buffer must have the same length of the the thermometer channels number.

Parameters

in	<i>ip</i>	pointer to a BaseThermometer class.
in	<i>sp</i>	pointer to a buffer of sensitivity values.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.49.2.10 `#define thermometerResetSensitivity(ip) (ip)->vmt->reset_sensitivity(ip)`

Reset thermometer sensitivity data restoring it to its typical value.

Parameters

in	<i>ip</i>	pointer to a BaseThermometer class.
----	-----------	---

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more errors occurred.

Function Class:

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

7.50 UART Driver

Generic UART Driver.

7.50.1 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.

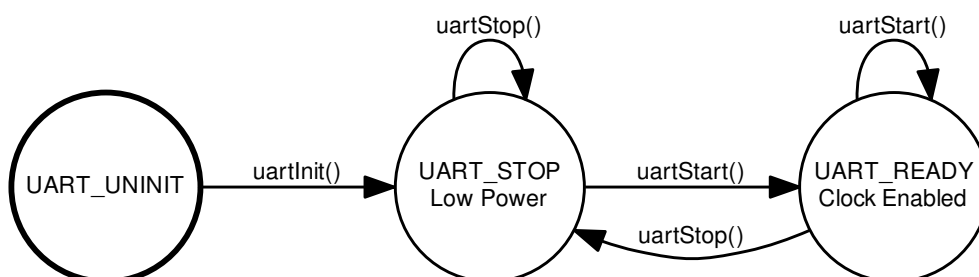
If your application requires a synchronous buffered driver then the [Serial Driver](#) should be used instead.

Precondition

In order to use the UART driver the `HAL_USE_UART` option must be enabled in `halconf.h`.

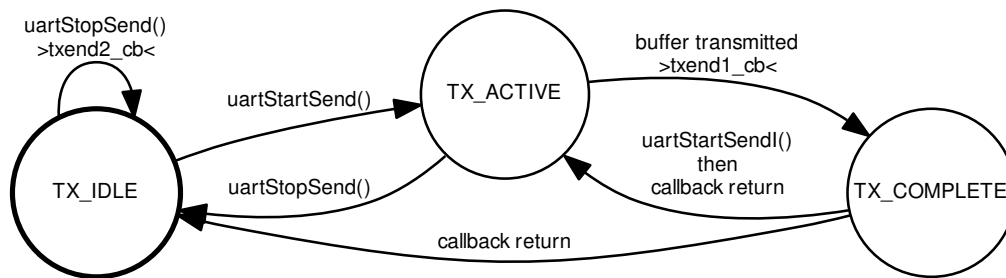
7.50.2 Driver State Machine

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).



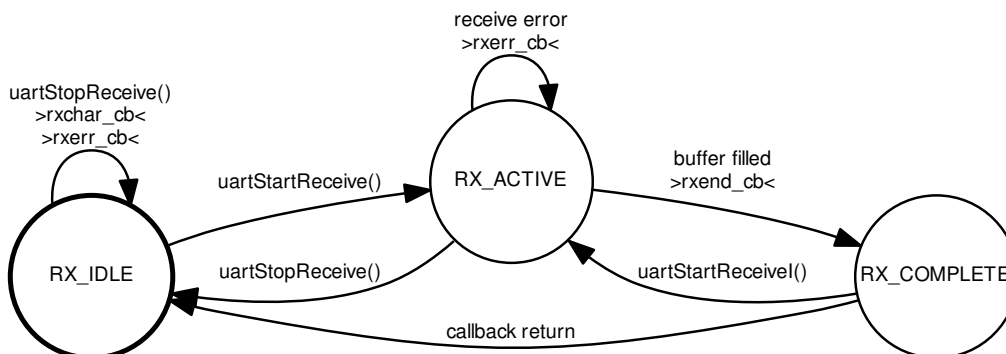
7.50.2.1 Transmitter sub State Machine

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.



7.50.2.2 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 error conditions

- #define `UART_ERR_NOT_ACTIVE` (size_t)-1

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_wakeup_rx_timeout_isr(uartp)`
- *Wakes up the waiting thread in case of RX timeout.*
- #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.*
- #define `_uart_timeout_isr_code(uartp)`
- *Timeout ISR code for receiver.*

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(* [uartecb_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 [uartStartSendI](#) ([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 [uartStopSendI](#) ([UARTDriver](#) *uartp)
Stops any ongoing transmission.
- void [uartStartReceive](#) ([UARTDriver](#) *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- void [uartStartReceiveI](#) ([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 [uartStopReceiveI](#) ([UARTDriver](#) *uartp)
Stops any ongoing receive operation.
- [msg_t](#) [uartSendTimeout](#) ([UARTDriver](#) *uartp, size_t *np, const void *txbuf, [sysinterval_t](#) timeout)
Performs a transmission on the UART peripheral.
- [msg_t](#) [uartSendFullTimeout](#) ([UARTDriver](#) *uartp, size_t *np, const void *txbuf, [sysinterval_t](#) timeout)
Performs a transmission on the UART peripheral.

- `msg_t uartReceiveTimeout (UARTDriver *uartp, size_t *np, void *rxbuf, sysinterval_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 *rxbuf)`
Starts a receive operation on the UART peripheral.
- `size_t uart_lld_stop_receive (UARTDriver *uartp)`
Stops any ongoing receive operation.

Enumerations

Variables

- `UARTDriver UARTD1`
UART1 driver identifier.

7.50.3 Macro Definition Documentation

7.50.3.1 `#define UART_NO_ERROR 0`

No pending conditions.

7.50.3.2 `#define UART_PARITY_ERROR 4`

Parity error happened.

7.50.3.3 `#define UART_FRAMING_ERROR 8`

Framing error happened.

7.50.3.4 `#define UART_OVERRUN_ERROR 16`

Overflow happened.

7.50.3.5 `#define UART_NOISE_ERROR 32`

Noise on the line.

7.50.3.6 #define UART_BREAK_DETECTED 64

Break detected.

7.50.3.7 #define UART_USE_WAIT FALSE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.50.3.8 #define UART_USE_MUTUAL_EXCLUSION FALSE

Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.50.3.9 #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	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
----	--------------------	---

Function Class:

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

7.50.3.10 #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 UARTDriver object
----	--------------	--

Function Class:

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

7.50.3.11 #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 UARTDriver object
----	--------------	--

Function Class:

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

7.50.3.12 #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 UARTDriver object
----	--------------	--

Function Class:

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

7.50.3.13 #define _uart_wakeup_rx_timeout_isr(*uartp*)**Value:**

```

{
    osalSysLockFromISR();
    osalThreadResumeI(&(uartp)->threadrx, MSG_TIMEOUT);
    osalSysUnlockFromISR();
}

```

Wakes up the waiting thread in case of RX timeout.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

Function Class:

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

7.50.3.14 #define _uart_tx1_isr_code(*uartp*)

Value:

```

{
    (uartp)->txstate = UART_TX_COMPLETE;
    if ((uartp)->config->txendl_cb != NULL) {
        (uartp)->config->txendl_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 UARTDriver object
----	--------------	--

Function Class:

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

7.50.3.15 #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 UARTDriver object
----	--------------	--

Function Class:

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

7.50.3.16 #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 UARTDriver object
----	--------------	--

Function Class:

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

7.50.3.17 #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 UARTDriver object
in	<i>errors</i>	mask of errors to be reported

Function Class:

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

7.50.3.18 #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 UARTDriver object
----	--------------	--

Function Class:

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

7.50.3.19 #define _uart_timeout_isr_code(*uartp*)**Value:**

```
{
    if ((uartp)->config->timeout_cb != NULL) {
        (uartp)->config->timeout_cb(uartp);
    }
    _uart_wakeup_rx_timeout_isr(uartp);
}
```

Timeout ISR code for receiver.

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 UARTDriver object
----	--------------	--

Function Class:

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

7.50.3.20 #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`.

7.50.4 Typedef Documentation**7.50.4.1 typedef uint32_t uartflags_t**

UART driver condition flags type.

7.50.4.2 typedef struct UARTDriver UARTDriver

Type of structure representing an UART driver.

7.50.4.3 typedef void(* uartcb_t) (UARTDriver *uartp)

Generic UART notification callback type.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

7.50.4.4 typedef void(* uartccb_t) (UARTDriver *uartp, uint16_t c)

Character received UART notification callback type.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object triggering the callback
in	<i>c</i>	received character

7.50.4.5 typedef void(* uartecb_t) (UARTDriver *uartp, uartflags_t e)

Receive error UART notification callback type.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object triggering the callback
in	<i>e</i>	receive error mask

7.50.5 Enumeration Type Documentation**7.50.5.1 enum uartstate_t**

Driver state machine possible states.

Enumerator

UART_UNINIT Not initialized.

UART_STOP Stopped.

UART_READY Ready.

7.50.5.2 enum uarttxstate_t

Transmitter state machine states.

Enumerator

UART_TX_IDLE Not transmitting.

UART_TX_ACTIVE Transmitting.

UART_TX_COMPLETE Buffer complete.

7.50.5.3 enum uartrxstate_t

Receiver state machine states.

Enumerator

UART_RX_IDLE Not receiving.

UART_RX_ACTIVE Receiving.

UART_RX_COMPLETE Buffer complete.

7.50.6 Function Documentation

7.50.6.1 void uartInit (void)

UART Driver initialization.

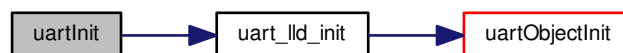
Note

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:



7.50.6.2 void uartObjectInit (UARTDriver * uartp)

Initializes the standard part of a `UARTDriver` structure.

Parameters

out	<i>uartp</i>	pointer to the UARTDriver 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:

**7.50.6.3 void uartStart (UARTDriver * *uartp*, const UARTConfig * *config*)**

Configures and activates the UART peripheral.

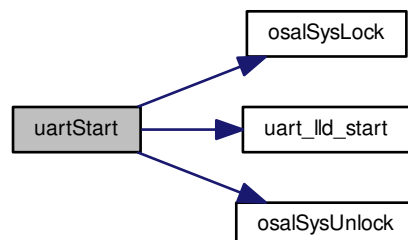
Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
in	<i>config</i>	pointer to the UARTConfig 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:



7.50.6.4 void uartStop (UARTDriver * uartp)

Deactivates the UART peripheral.

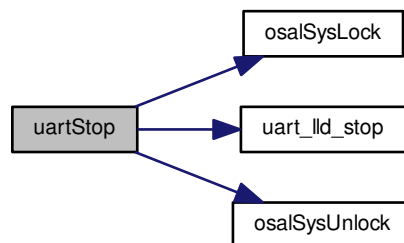
Parameters

in	<i>uartp</i>	pointer to the UARTDriver 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:



7.50.6.5 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.

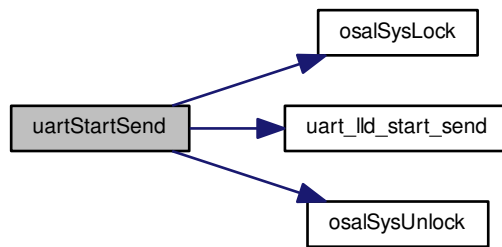
Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
in	<i>n</i>	number of data frames 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:



7.50.6.6 `void uartStartSendI (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:



7.50.6.7 `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 UARTDriver object
----	--------------	--

Returns

The number of data frames not transmitted by the stopped transmit operation.

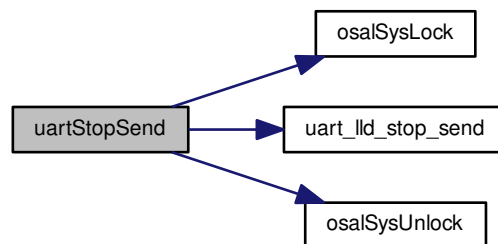
Return values

<code>UART_ERR_NOT_ACTIVE</code>	if 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:

**7.50.6.8 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 UARTDriver object
----	--------------	--

Returns

The number of data frames not transmitted by the stopped transmit operation.

Return values

<code>UART_ERR_NOT_ACTIVE</code>	if 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:



7.50.6.9 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.

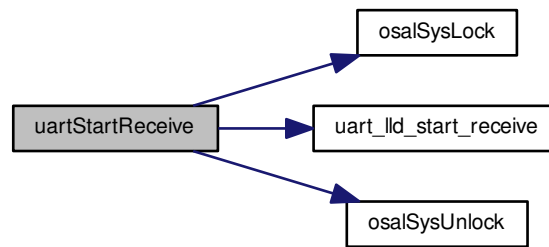
Parameters

in	<i>uartp</i>	pointer to the UARTDriver 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:



7.50.6.10 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.

This function has to be invoked from a lock zone.

Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
in	<code>n</code>	number of data frames to receive
out	<code>rxbuf</code>	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:



7.50.6.11 `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 UARTDriver object
----	--------------	--

Returns

The number of data frames not received by the stopped receive operation.

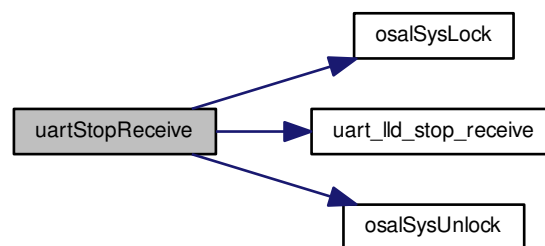
Return values

<code>UART_ERR_NOT_ACTIVE</code>	if 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:

**7.50.6.12 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	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

Returns

The number of data frames not received by the stopped receive operation.

Return values

<code>UART_ERR_NOT_ACTIVE</code>	if 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:



7.50.6.13 `msg_t uartSendTimeout (UARTDriver * uartp, size_t * np, const void * txbuf, sysinterval_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. This function implements a software timeout, it does not use any underlying HW timeout mechanism.

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	<code>txbuf</code>	the pointer to the transmit buffer
in	<code>timeout</code>	operation timeout

Returns

The operation status.

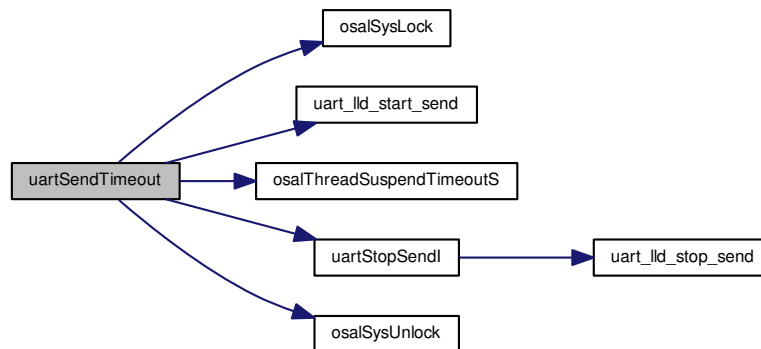
Return values

<code>MSG_OK</code>	if the operation completed successfully.
<code>MSG_TIMEOUT</code>	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:



7.50.6.14 `msg_t uartSendFullTimeout (UARTDriver * uartp, size_t * np, const void * txbuf, sysinterval_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.

This function implements a software timeout, it does not use any underlying HW timeout mechanism.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver 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.

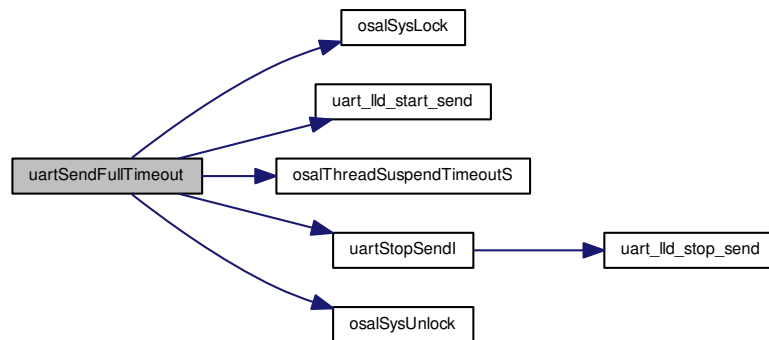
Return values

<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:



7.50.6.15 `msg_t uartReceiveTimeout (UARTDriver * uartp, size_t * np, void * rxbuf, sysinterval_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.
 This function implements a software timeout, it does not use any underlying HW timeout mechanism.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver 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.

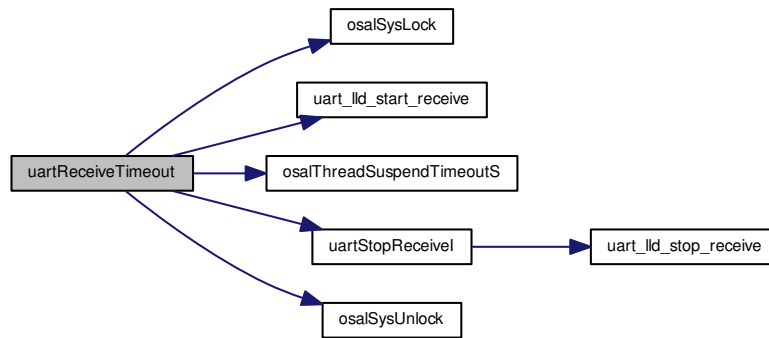
Return values

<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:



7.50.6.16 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 <code>UARTDriver</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:



7.50.6.17 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	<i>uartp</i>	pointer to the UARTDriver 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:

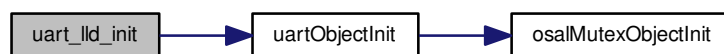
**7.50.6.18 void uart_lld_init (void)**

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:

**7.50.6.19 void uart_lld_start (UARTDriver * uartp)**

Configures and activates the UART peripheral.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

Function Class:

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

7.50.6.20 void `uart_lld_stop (UARTDriver * uartp)`

Deactivates the UART peripheral.

Parameters

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
----	--------------	---

Function Class:

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

7.50.6.21 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 <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:

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

7.50.6.22 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 <code>UARTDriver</code> 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.

7.50.6.23 `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 UARTDriver 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.

7.50.6.24 `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 UARTDriver 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.

7.50.7 Variable Documentation**7.50.7.1 UARTDriver UARTD1**

UART1 driver identifier.

7.51 USB Driver

Generic USB Driver.

7.51.1 Detailed Description

Generic USB Driver.

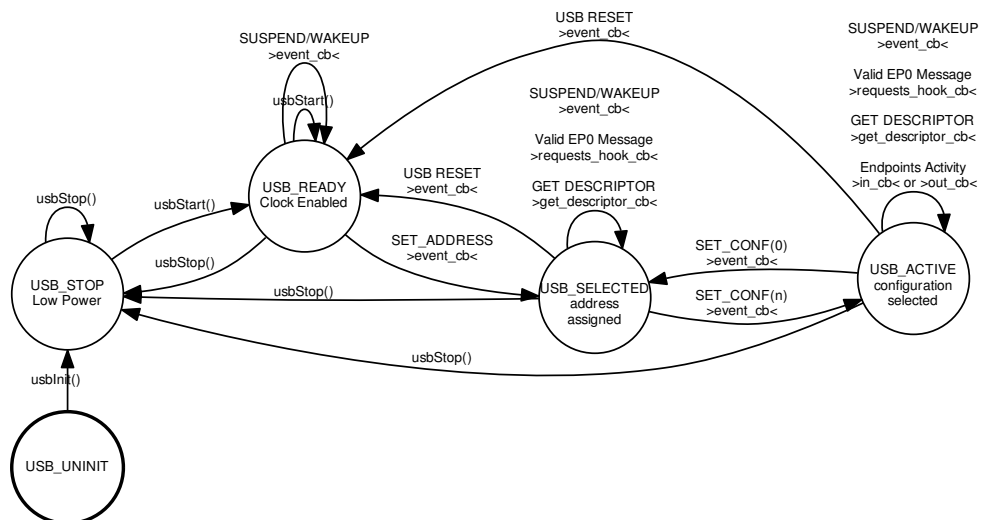
This module implements a generic USB (Universal Serial Bus) driver supporting device-mode operations.

Precondition

In order to use the USB driver the `HAL_USE_USB` option must be enabled in `halconf.h`.

7.51.2 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).



7.51.3 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.

7.51.3.1 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

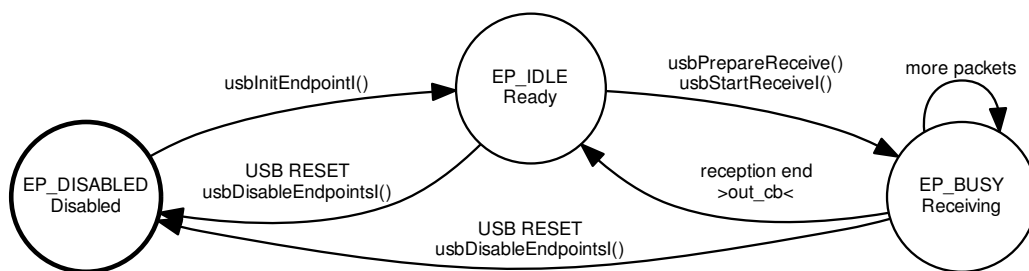
required by the USB device class to be implemented and of the handling of the specific messages sent over the endpoint zero. Standard messages are handled internally to the driver. The application can use hooks in order to handle custom messages or override the handling of the default handling of standard messages.

7.51.3.2 USB Endpoints

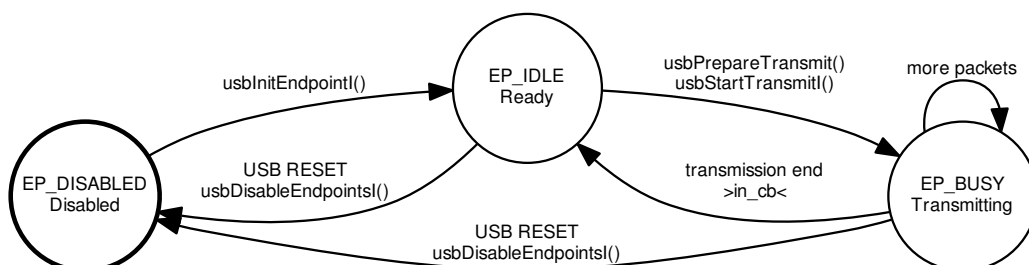
USB endpoints are the objects that the application uses to exchange data with the host. There are two kind of endpoints:

- **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:



7.51.3.3 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_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.
- `#define usb_lld_wakeup_host(usbp)`
Start of host wake-up procedure.

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, bMaxPacket←
Size, 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 `usbGetDriverStatel`(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 `usbGetTransmitStatusl`(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)
- Returns the status of an IN endpoint.*
- #define `usbGetReceiveStatusl`(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(* 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.

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 `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`)
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 `usbStallReceiveI` (`USBDriver *usbp`, `usbep_t ep`)
Stalls an OUT endpoint.
- bool `usbStallTransmitI` (`USBDriver *usbp`, `usbep_t ep`)
Stalls an IN endpoint.
- void `usbWakeupHost` (`USBDriver *usbp`)
Host wake-up procedure.
- 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

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.

7.51.4 Macro Definition Documentation

7.51.4.1 #define USB_DESC_INDEX(i) ((uint8_t)(i))

Helper macro for index values into descriptor strings.

7.51.4.2 #define USB_DESC_BYTE(b) ((uint8_t)(b))

Helper macro for byte values into descriptor strings.

7.51.4.3 #define USB_DESC_WORD(w)

Value:

```
(uint8_t)((w) & 255U),  
(uint8_t)((w) >> 8) & 255U) \
```

Helper macro for word values into descriptor strings.

7.51.4.4 #define USB_DESC_BCD(bcd)

Value:

```
(uint8_t)((bcd) & 255U),  
(uint8_t)((bcd) >> 8) & 255) \
```

Helper macro for BCD values into descriptor strings.

7.51.4.5 #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.

7.51.4.6 #define USB_DESC_CONFIGURATION_SIZE 9U

Configuration Descriptor size.

7.51.4.7 **#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.

7.51.4.8 **#define** USB_DESC_INTERFACE_SIZE 9U

Interface Descriptor size.

7.51.4.9 **#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.

7.51.4.10 **#define** USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U

Interface Association Descriptor size.

7.51.4.11 **#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.

7.51.4.12 #define USB_DESC_ENDPOINT_SIZE 7U

Endpoint Descriptor size.

7.51.4.13 #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.

7.51.4.14 #define USB_EP_MODE_TYPE 0x0003U

Endpoint type mask.

7.51.4.15 #define USB_EP_MODE_TYPE_CTRL 0x0000U

Control endpoint.

7.51.4.16 #define USB_EP_MODE_TYPE_ISOC 0x0001U

Isochronous endpoint.

7.51.4.17 #define USB_EP_MODE_TYPE_BULK 0x0002U

Bulk endpoint.

7.51.4.18 #define USB_EP_MODE_TYPE_INTR 0x0003U

Interrupt endpoint.

7.51.4.19 #define USB_USE_WAIT FALSE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.51.4.20 #define usbGetDriverState(*usbp*) ((usbp)->state)

Returns the driver state.

Parameters

in	<i>usbp</i>	pointer to the USBDriver 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.

7.51.4.21 #define usbConnectBus(*usbp*) usb_lld_connect_bus(usbp)

Connects the USB device.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.51.4.22 #define usbDisconnectBus(*usbp*) usb_lld_disconnect_bus(usbp)

Disconnect the USB device.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.51.4.23 #define usbGetFrameNumberX(*usbp*) usb_lld_get_frame_number(usbp)

Returns the current frame number.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Returns

The current frame number.

Function Class:

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

```
7.51.4.24 #define usbGetTransmitStatus( 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 USBDriver 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.

```
7.51.4.25 #define usbGetReceiveStatus( 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 USBDriver 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.

7.51.4.26 `#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 <code>USBDriver</code> 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.

7.51.4.27 `#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 <code>USBDriver</code> 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 <code>NULL</code>

Function Class:

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

7.51.4.28 #define usbReadSetup(*usbp*, *ep*, *buf*) usb_ll_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 USBDriver 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.

7.51.4.29 #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 USBDriver object
in	<i>evt</i>	USB event code

Function Class:

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

7.51.4.30 #define _usb_isr_invoke_sof_cb(*usbp*)

Value:

```
{
    if ((usbp->config->sof_cb) != NULL) {
        (usbp->config->sof_cb)(usbp);
    }
}
```

Common ISR code, SOF callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.51.4.31 #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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.4.32 #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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.4.33 `#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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.4.34 `#define USB_MAX_ENDPOINTS 4`

Maximum endpoint address.

7.51.4.35 `#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW`

Status stage handling method.

7.51.4.36 `#define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS`

The address can be changed immediately upon packet reception.

7.51.4.37 `#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW`

Method for set address acknowledge.

7.51.4.38 `#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`.

7.51.4.39 `#define usb_lld_get_frame_number(usbp) 0`

Returns the current frame number.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Returns

The current frame number.

Function Class:

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

7.51.4.40 `#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 USBDriver object
in	<i>ep</i>	endpoint number

Returns

Received data size.

Function Class:

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

7.51.4.41 `#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.

7.51.4.42 `#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.

7.51.4.43 `#define usb_lld_wakeup_host(usbp)`

Start of host wake-up procedure.

Function Class:

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

7.51.5 Typedef Documentation

7.51.5.1 `typedef struct USBDriver USBDriver`

Type of a structure representing an USB driver.

7.51.5.2 `typedef uint8_t usbep_t`

Type of an endpoint identifier.

7.51.5.3 `typedef void(* usbcallback_t) (USBDriver *usbp)`

Type of an USB generic notification callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object triggering the callback
----	-------------	---

7.51.5.4 `typedef void(* usbepcallback_t) (USBDriver *usbp, usbep_t ep)`

Type of an USB endpoint callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object triggering the callback
in	<i>ep</i>	endpoint number

7.51.5.5 `typedef void(* usbeventcb_t) (USBDriver *usbp, usbevent_t event)`

Type of an USB event notification callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object triggering the callback
in	<i>event</i>	event type

7.51.5.6 `typedef bool(* usbreqhandler_t) (USBDriver *usbp)`

Type of a requests handler callback.

The request is encoded in the `usb_setup` buffer.

Parameters

<i>in</i>	<i>usbp</i>	pointer to the USBDriver 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.

7.51.5.7 `typedef const USBDescriptor*(* usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`

Type of an USB descriptor-retrieving callback.

7.51.6 Enumeration Type Documentation

7.51.6.1 `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.

7.51.6.2 `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.

7.51.6.3 `enum usbep0state_t`

Type of an endpoint zero state machine states.

Enumerator

USB_EP0_STP_WAITING Waiting for SETUP data.

USB_EP0_IN_TX Transmitting.
USB_EP0_IN_WAITING_TX0 Waiting transmit 0.
USB_EP0_IN_SENDING_STS Sending status.
USB_EP0_OUT_WAITING_STS Waiting status.
USB_EP0_OUT_RX Receiving.
USB_EP0_ERROR Error, EP0 stalled.

7.51.6.4 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_UNCONFIGURED Configuration removed.
USB_EVENT_SUSPEND Entering suspend mode.
USB_EVENT_WAKEUP Leaving suspend mode.
USB_EVENT_STALLED Endpoint 0 error, stalled.

7.51.7 Function Documentation

7.51.7.1 static void set_address (USBDriver * usbp) [static]

SET ADDRESS transaction callback.

Parameters

in	usbp	pointer to the USBDriver object
----	------	---

Here is the call graph for this function:



7.51.7.2 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 USBDriver object
----	-------------	---

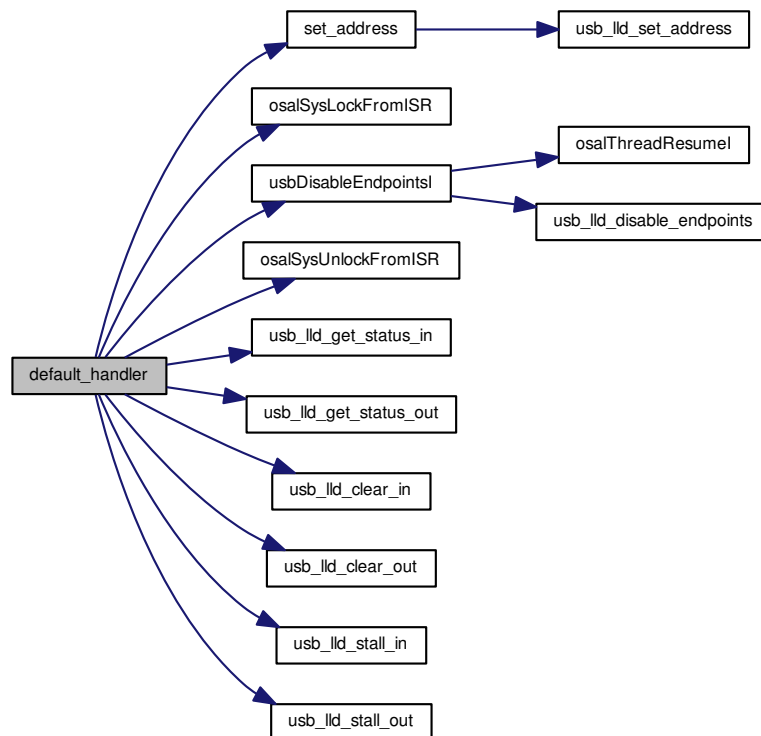
Returns

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:



7.51.7.3 void usbInit (void)

USB Driver initialization.

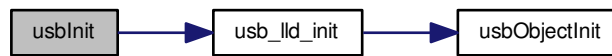
Note

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:



7.51.7.4 void usbObjectInit (USBDriver * *usbp*)

Initializes the standard part of a `USBDriver` structure.

Parameters

out	<i>usbp</i>	pointer to the <code>USBDriver</code> object
-----	-------------	--

Function Class:

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

7.51.7.5 void usbStart (USBDriver * *usbp*, const USBConfig * *config*)

Configures and activates the USB peripheral.

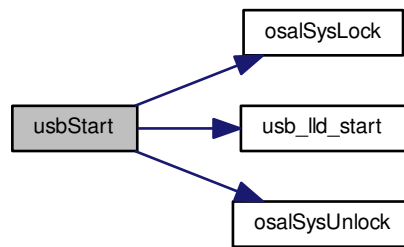
Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>config</i>	pointer to the <code>USBConfig</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:



7.51.7.6 void usbStop (USBDriver * usbp)

Deactivates the USB peripheral.

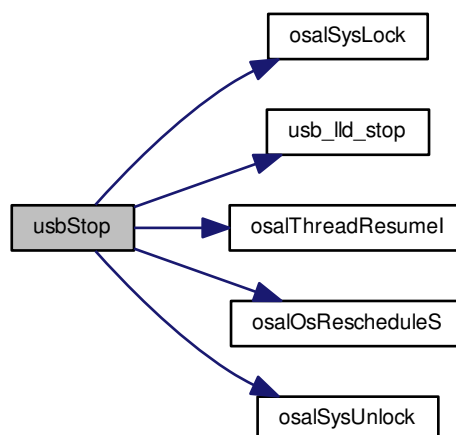
Parameters

in	<i>usbp</i>	pointer to the USBDriver 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:



7.51.7.7 void usbInitEndpointI (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 USBDriver 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:



7.51.7.8 void usbDisableEndpointsI (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.

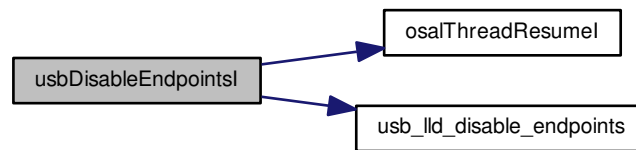
Parameters

in	<i>usbp</i>	pointer to the USBDriver 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:



7.51.7.9 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 USBDriver 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:



7.51.7.10 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 USBDriver 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:



7.51.7.11 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 USBDriver 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.

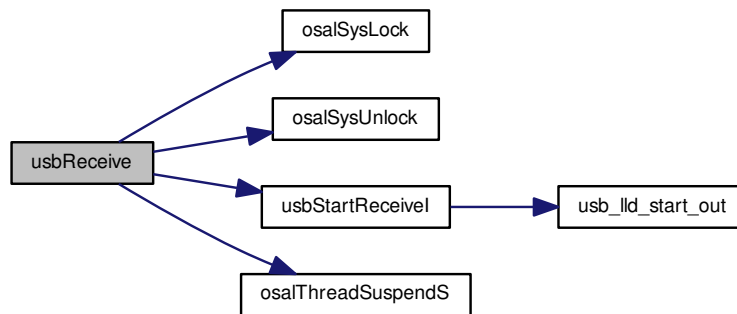
Return values

MSG_RESET	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:



7.51.7.12 `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 USBDriver 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.

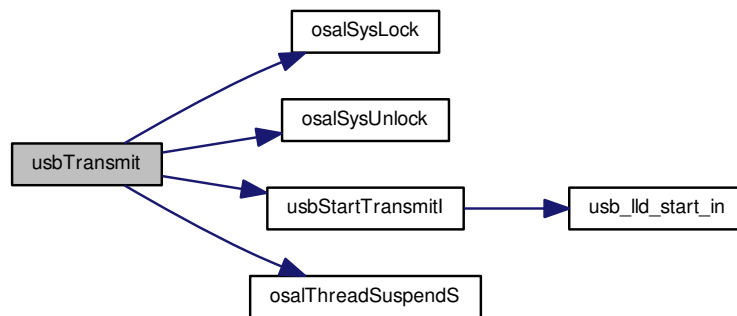
Return values

<i>MSG_OK</i>	operation performed successfully.
<i>MSG_RESET</i>	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:



7.51.7.13 bool usbStallReceive(USBDriver * usbp, usbep_t ep)

Stalls an OUT endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver 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:



7.51.7.14 `bool usbStallTransmitl (USBDriver * usbp, usbep_t ep)`

Stalls an IN endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver 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:

**7.51.7.15** `void usbWakeupHost (USBDriver * usbp)`

Host wake-up procedure.

Note

It is silently ignored if the USB device is not in the `USB_SUSPENDED` state.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.51.7.16 void _usb_reset (USBDriver * usbp)

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.

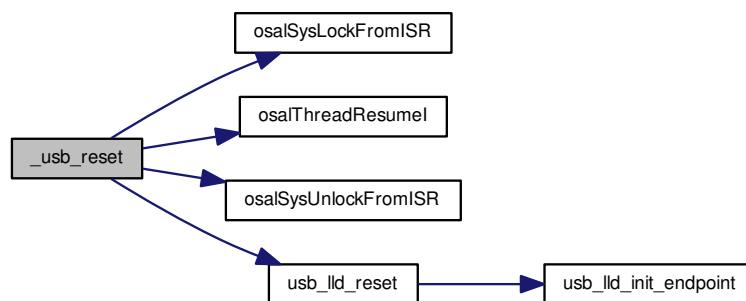
Parameters

in	usbp	pointer to the USBDriver object
----	------	---

Function Class:

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

Here is the call graph for this function:



7.51.7.17 void _usb_suspend (USBDriver * usbp)

USB suspend routine.

This function must be invoked when an USB bus suspend condition is detected.

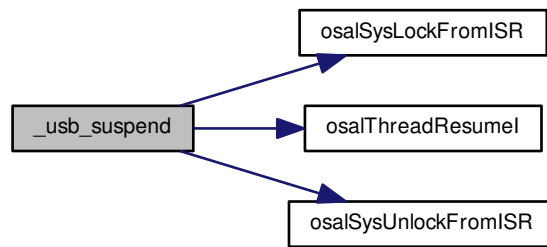
Parameters

in	usbp	pointer to the USBDriver object
----	------	---

Function Class:

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

Here is the call graph for this function:



7.51.7.18 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 USBDriver object
----	-------------	---

Function Class:

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

7.51.7.19 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.

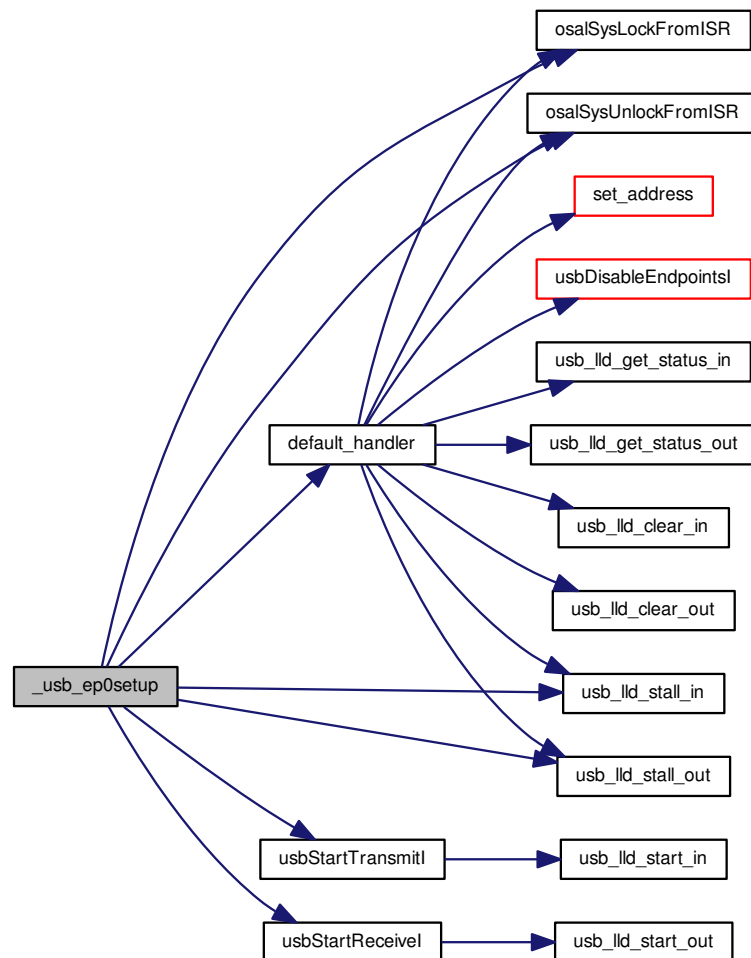
Parameters

in	<i>usbp</i>	pointer to the USBDriver 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:



7.51.7.20 `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.

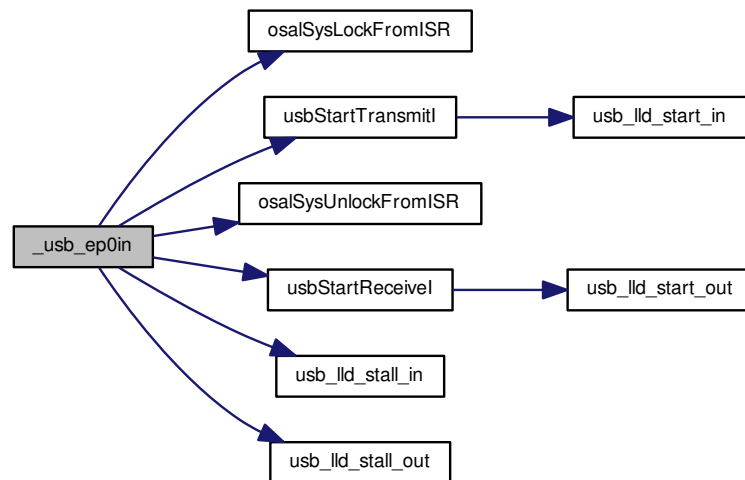
Parameters

in	<i>usbp</i>	pointer to the USBDriver 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:



7.51.7.21 `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.

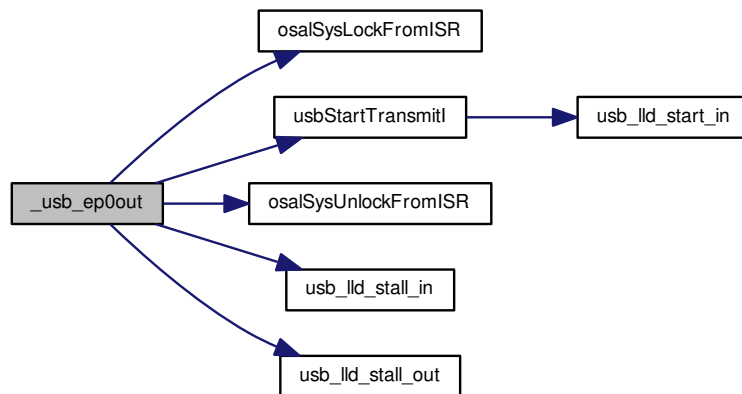
Parameters

in	<i>usbp</i>	pointer to the USBDriver 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:



7.51.7.22 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:



7.51.7.23 void usb_lld_start (USBDriver * usbp)

Configures and activates the USB peripheral.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.51.7.24 void usb_lld_stop (USBDriver * usbp)

Deactivates the USB peripheral.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.51.7.25 void usb_lld_reset (USBDriver * usbp)

USB low level reset routine.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:

**7.51.7.26 void usb_lld_set_address (USBDriver * usbp)**

Sets the USB address.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.51.7.27 void usb_lld_init_endpoint (USBDriver * usbp, usbep_t ep)

Enables an endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.7.28 void usb_lld_disable_endpoints (USBDriver * *usbp*)

Disables all the active endpoints except the endpoint zero.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.51.7.29 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 USBDriver 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.

7.51.7.30 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 USBDriver 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.

7.51.7.31 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 USBDriver 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.

7.51.7.32 void `usb_lld_prepare_receive (USBDriver * usbp, usbep_t ep)`

Prepares for a receive operation.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.7.33 void usb_lld_prepare_transmit (USBDriver * *usbp*, *usbep_t ep*)

Prepares for a transmit operation.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.7.34 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.7.35 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.7.36 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.7.37 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.7.38 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.7.39 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.51.8 Variable Documentation

7.51.8.1 USBDriver USBD1

USB1 driver identifier.

7.51.8.2 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.

7.51.8.3 USBInEndpointState { ... } in

IN EP0 state.

7.51.8.4 USBOutEndpointState { ... } out

OUT EP0 state.

7.51.8.5 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.

7.52 USB CDC Header

USB CDC Support Header.

7.52.1 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.

7.53 WDG Driver

Generic WDG Driver.

7.53.1 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)`

Deactivates the WDG peripheral.

- void `wdg_lld_reset` (`WDGDriver` *wdgp)

Reloads WDG's counter.

Enumerations

7.53.2 Macro Definition Documentation

7.53.2.1 `#define wdgResetl(wdgp) wdg_lld_reset(wdgp)`

Resets WDG's counter.

Parameters

in	wdgp	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.

7.53.2.2 `#define PLATFORM_WDG_USE_WDG1 FALSE`

WDG1 driver enable switch.

Note

The default is `FALSE`.

7.53.3 Typedef Documentation

7.53.3.1 `typedef struct WDGDriver WDGDriver`

Type of a structure representing an WDG driver.

7.53.4 Enumeration Type Documentation

7.53.4.1 `enum wdgstate_t`

Driver state machine possible states.

Enumerator

`WDG_UNINIT` Not initialized.

`WDG_STOP` Stopped.

`WDG_READY` Ready.

7.53.5 Function Documentation

7.53.5.1 `void wdgInit(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:



7.53.5.2 void wdgStart (WDGDriver * wdgp, const WDGConfig * config)

Configures and activates the WDG peripheral.

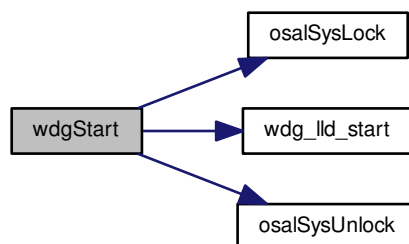
Parameters

in	<i>wdgp</i>	pointer to the <code>WDGDriver</code> object
in	<i>config</i>	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:



7.53.5.3 void wdgStop (WDGDriver * wdgp)

Deactivates the WDG peripheral.

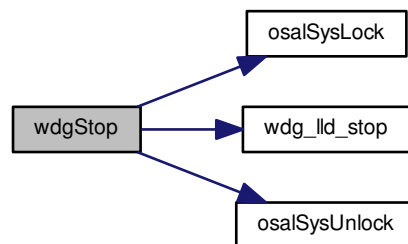
Parameters

in	<i>wdgp</i>	pointer to the WDGDriver 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:

**7.53.5.4 void wdgReset (WDGDriver * wdgp)**

Resets WDG's counter.

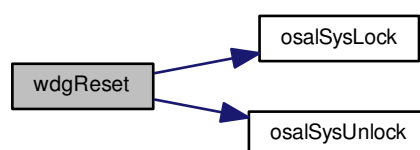
Parameters

in	<i>wdgp</i>	pointer to the WDGDriver 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:



7.53.5.5 void wdg_llc_init (void)

Low level WDG driver initialization.

Function Class:

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

7.53.5.6 void wdg_llc_start (WDGDriver * wdgp)

Configures and activates the WDG peripheral.

Parameters

in	<i>wdgp</i>	pointer to the WDGDriver object
----	-------------	---

Function Class:

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

7.53.5.7 void wdg_llc_stop (WDGDriver * wdgp)

Deactivates the WDG peripheral.

Parameters

in	<i>wdgp</i>	pointer to the WDGDriver object
----	-------------	---

Function Class:

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

7.53.5.8 void wdg_llc_reset (WDGDriver * wdgp)

Reloads WDG's counter.

Parameters

in	<i>wdgp</i>	pointer to the WDGDriver object
----	-------------	---

Function Class:

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

Chapter 8

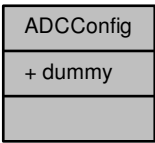
Data Structure Documentation

8.1 ADCConfig Struct Reference

Driver configuration structure.

```
#include <hal_adc_lld.h>
```

Collaboration diagram for ADCConfig:



8.1.1 Detailed Description

Driver configuration structure.

Note

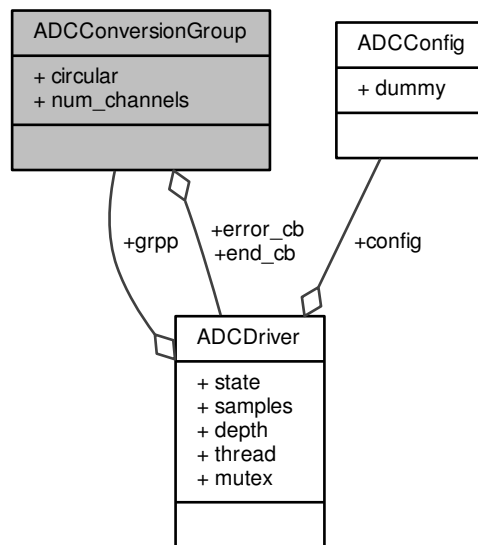
It could be empty on some architectures.

8.2 ADCConversionGroup Struct Reference

Conversion group configuration structure.

```
#include <hal_adc_llld.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 ADCCConversionGroup::circular`

Enables the circular buffer mode for the group.

8.2.2.2 `adc_channels_num_t ADCCConversionGroup::num_channels`

Number of the analog channels belonging to the conversion group.

8.2.2.3 `adccallback_t ADCCConversionGroup::end_cb`

Callback function associated to the group or `NULL`.

8.2.2.4 `adcerrorcallback_t ADCCConversionGroup::error_cb`

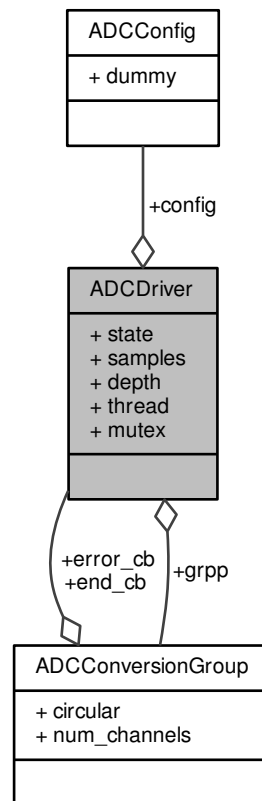
Error callback or `NULL`.

8.3 ADCDriver Struct Reference

Structure representing an ADC driver.

```
#include <hal_adc_llid.h>
```

Collaboration diagram for ADCDriver:



Data Fields

- `adcstate_t state`
Driver state.
- `const ADCCConfig * config`
Current configuration data.
- `adcsample_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`

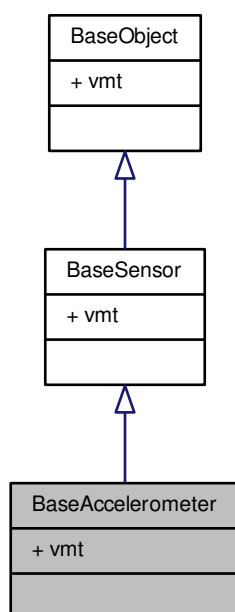
Mutex protecting the peripheral.

8.4 BaseAccelerometer Struct Reference

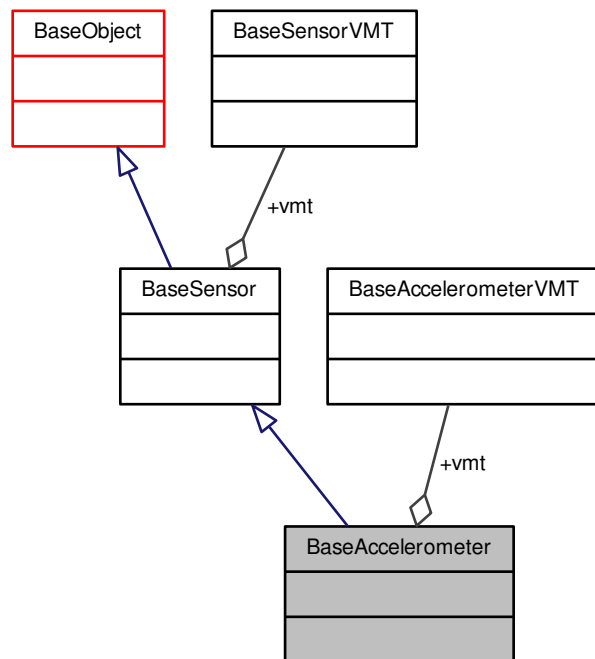
Base accelerometer class.

```
#include <hal_accelerometer.h>
```

Inheritance diagram for BaseAccelerometer:



Collaboration diagram for BaseAccelerometer:



Data Fields

- const struct [BaseAccelerometerVMT](#) * **vmt**

Virtual Methods Table.

8.4.1 Detailed Description

Base accelerometer class.

This class represents a generic a generic accelerometer.

8.4.2 Field Documentation

8.4.2.1 const struct **BaseAccelerometerVMT*** **BaseAccelerometer::vmt**

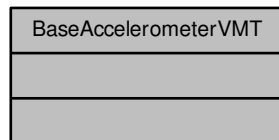
Virtual Methods Table.

8.5 BaseAccelerometerVMT Struct Reference

[BaseAccelerometer](#) virtual methods table.

```
#include <hal_accelerometer.h>
```

Collaboration diagram for BaseAccelerometerVMT:



8.5.1 Detailed Description

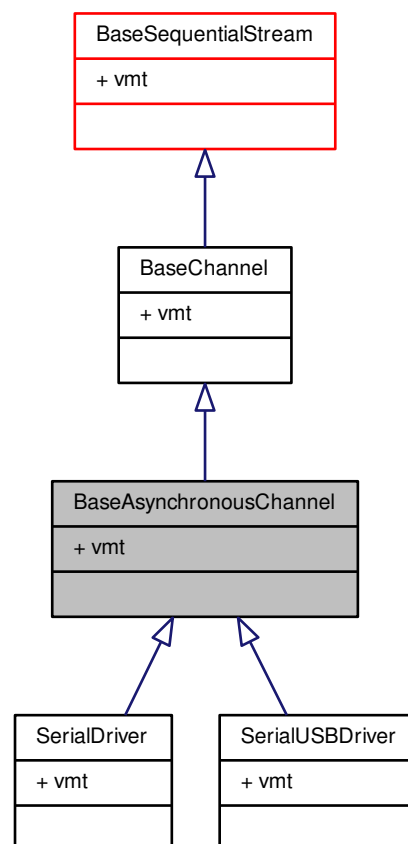
[BaseAccelerometer](#) virtual methods table.

8.6 BaseAsynchronousChannel Struct Reference

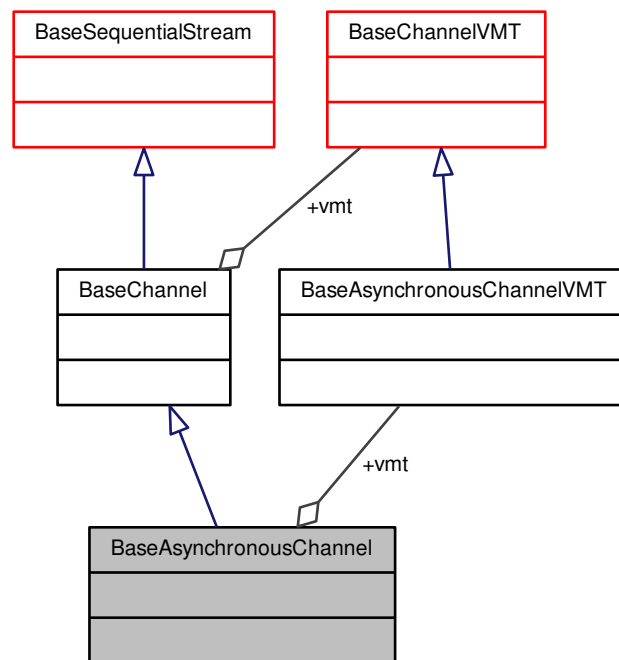
Base asynchronous channel class.

```
#include <hal_channels.h>
```


Inheritance diagram for BaseAsynchronousChannel:



Collaboration diagram for BaseAsynchronousChannel:



Data Fields

- const struct [BaseAsynchronousChannelVMT](#) * vmt

Virtual Methods Table.

8.6.1 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.

8.6.2 Field Documentation

8.6.2.1 const struct [BaseAsynchronousChannelVMT](#)* [BaseAsynchronousChannel](#)::vmt

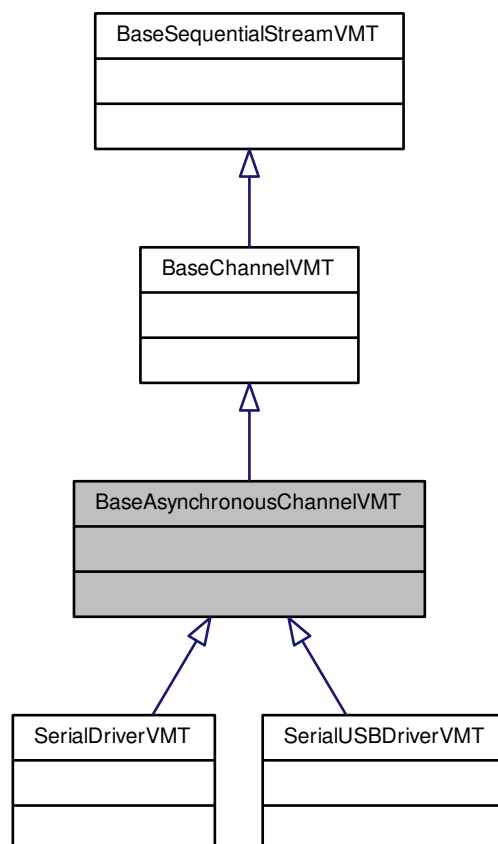
Virtual Methods Table.

8.7 BaseAsynchronousChannelVMT Struct Reference

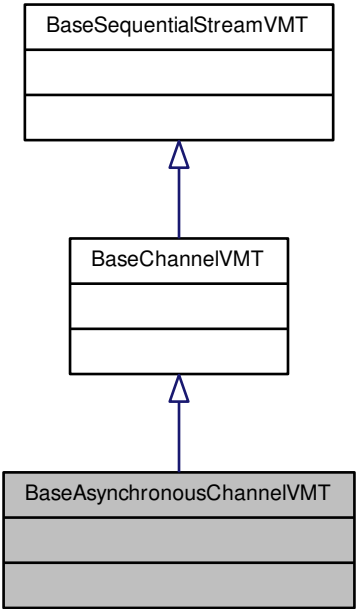
[BaseAsynchronousChannel](#) virtual methods table.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseAsynchronousChannelVMT:



Collaboration diagram for BaseAsynchronousChannelVMT:



8.7.1 Detailed Description

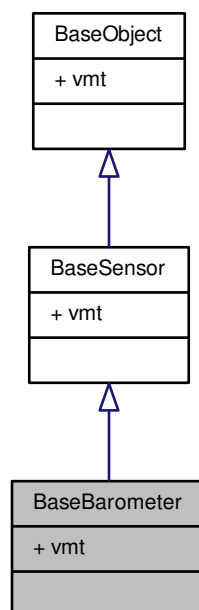
BaseAsynchronousChannel virtual methods table.

8.8 BaseBarometer Struct Reference

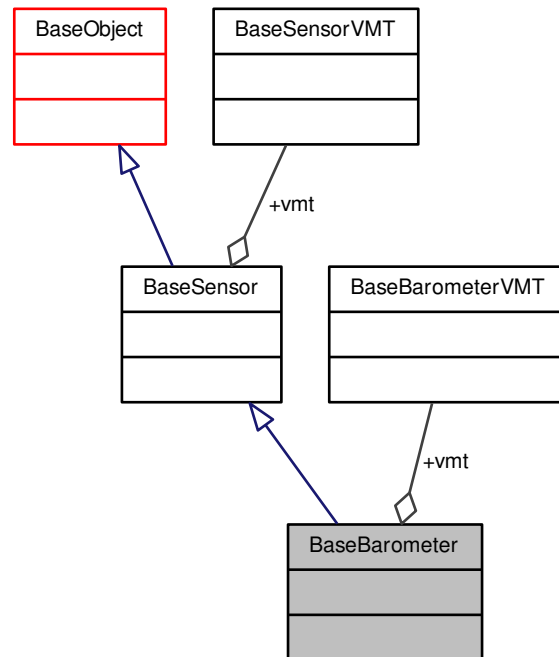
Base barometer class.

```
#include <hal_barometer.h>
```

Inheritance diagram for BaseBarometer:



Collaboration diagram for BaseBarometer:



Data Fields

- const struct [BaseBarometerVMT](#) * vmt

Virtual Methods Table.

8.8.1 Detailed Description

Base barometer class.

This class represents a generic barometer.

8.8.2 Field Documentation

8.8.2.1 const struct [BaseBarometerVMT](#)* BaseBarometer::vmt

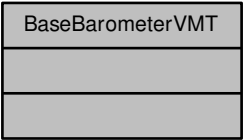
Virtual Methods Table.

8.9 BaseBarometerVMT Struct Reference

[BaseBarometer](#) virtual methods table.

```
#include <hal_barometer.h>
```

Collaboration diagram for BaseBarometerVMT:



8.9.1 Detailed Description

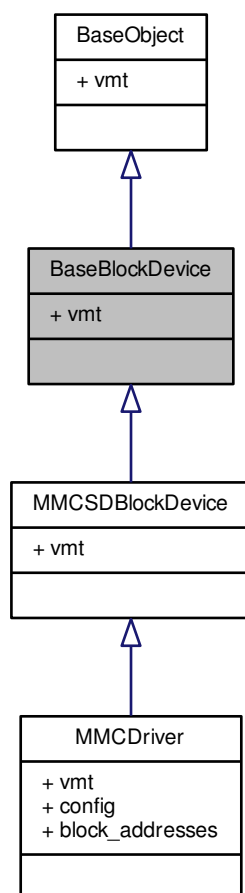
BaseBarometer virtual methods table.

8.10 BaseBlockDevice Struct Reference

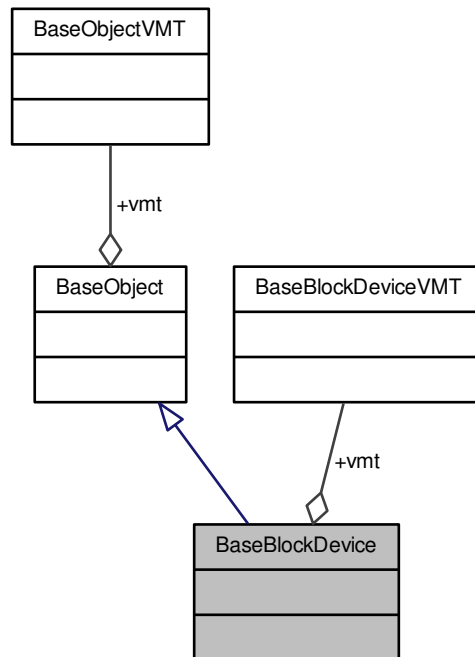
Base block device class.

```
#include <hal_ioblock.h>
```

Inheritance diagram for BaseBlockDevice:



Collaboration diagram for BaseBlockDevice:



Data Fields

- const struct [BaseBlockDeviceVMT](#) * vmt

Virtual Methods Table.

8.10.1 Detailed Description

Base block device class.

This class represents a generic, block-accessible, device.

8.10.2 Field Documentation

8.10.2.1 const struct [BaseBlockDeviceVMT](#)* BaseBlockDevice::vmt

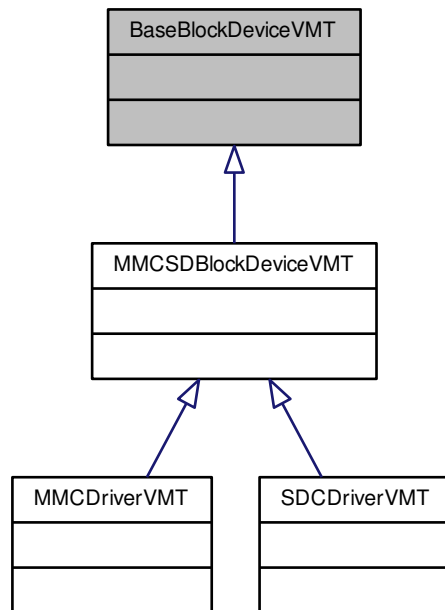
Virtual Methods Table.

8.11 BaseBlockDeviceVMT Struct Reference

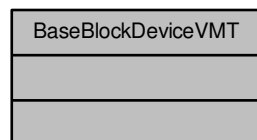
[BaseBlockDevice](#) virtual methods table.

```
#include <hal_ioblock.h>
```

Inheritance diagram for BaseBlockDeviceVMT:



Collaboration diagram for BaseBlockDeviceVMT:



8.11.1 Detailed Description

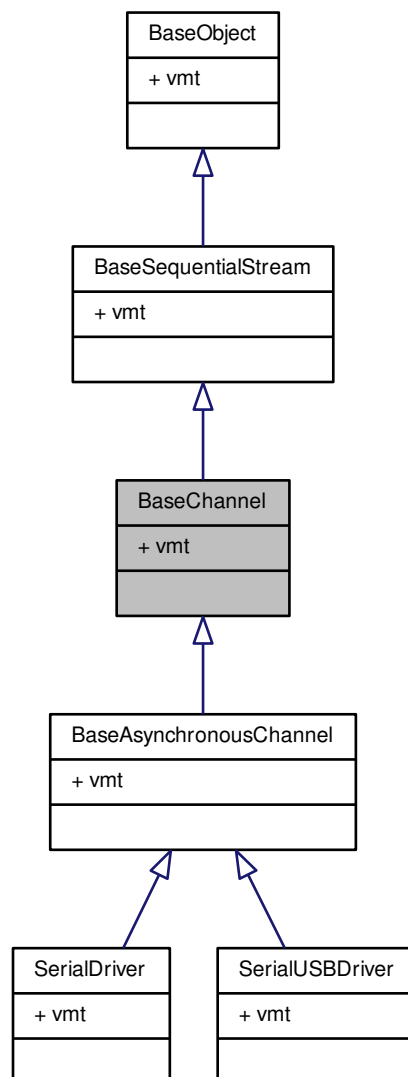
[BaseBlockDevice](#) virtual methods table.

8.12 BaseChannel Struct Reference

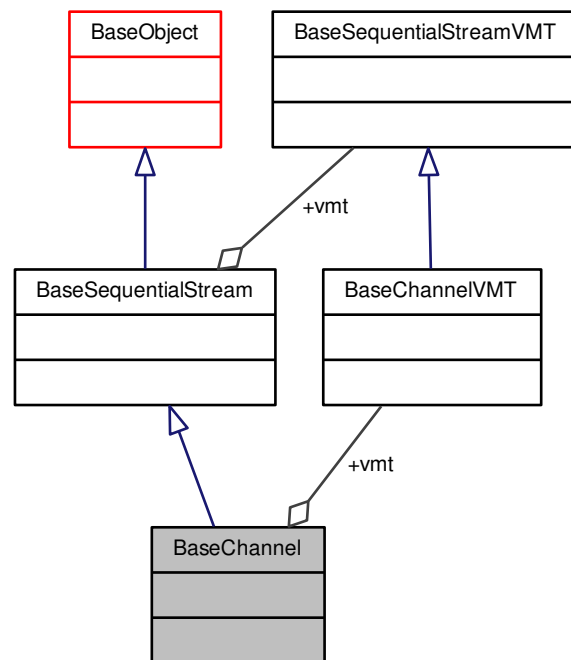
Base channel class.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseChannel:



Collaboration diagram for BaseChannel:



Data Fields

- const struct [BaseChannelVMT](#) * `vmt`

Virtual Methods Table.

8.12.1 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.

8.12.2 Field Documentation

8.12.2.1 const struct [BaseChannelVMT](#) * `BaseChannel::vmt`

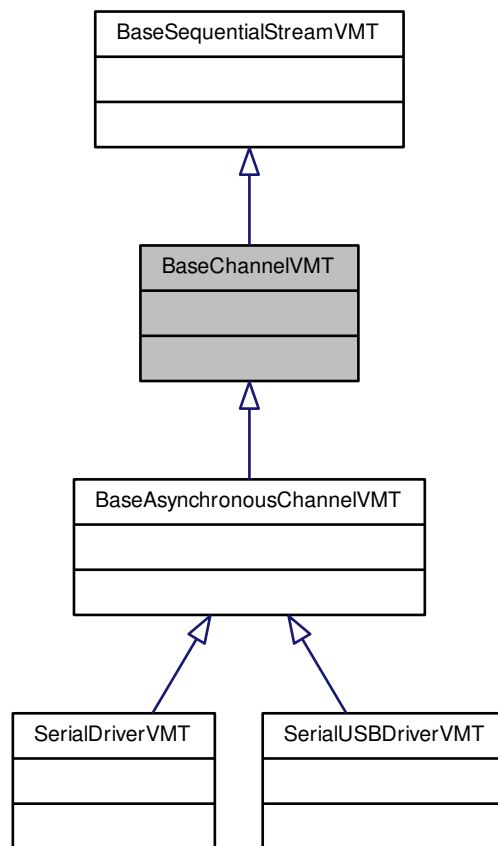
Virtual Methods Table.

8.13 BaseChannelVMT Struct Reference

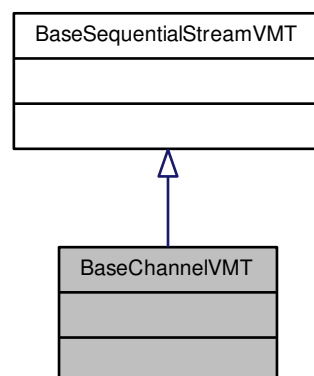
[BaseChannel](#) virtual methods table.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseChannelVMT:



Collaboration diagram for BaseChannelVMT:



8.13.1 Detailed Description

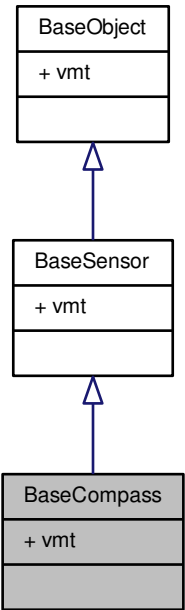
BaseChannel virtual methods table.

8.14 BaseCompass Struct Reference

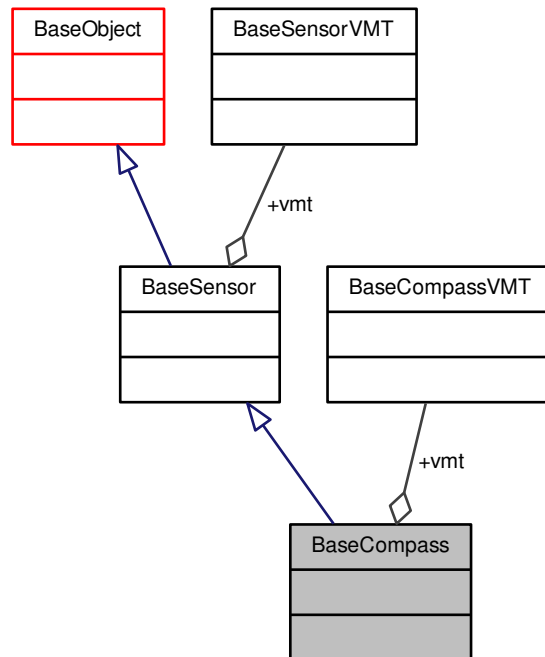
Base compass class.

```
#include <hal_compass.h>
```

Inheritance diagram for BaseCompass:



Collaboration diagram for BaseCompass:



Data Fields

- const struct [BaseCompassVMT](#) * vmt

Virtual Methods Table.

8.14.1 Detailed Description

Base compass class.

This class represents a generic compass.

8.14.2 Field Documentation

8.14.2.1 const struct [BaseCompassVMT](#)* BaseCompass::vmt

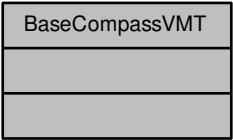
Virtual Methods Table.

8.15 BaseCompassVMT Struct Reference

[BaseCompass](#) virtual methods table.

```
#include <hal_compass.h>
```

Collaboration diagram for BaseCompassVMT:



8.15.1 Detailed Description

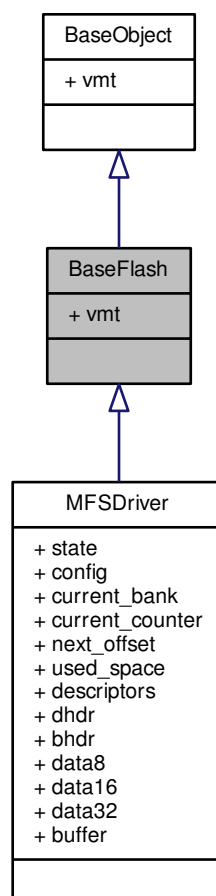
BaseCompass virtual methods table.

8.16 BaseFlash Struct Reference

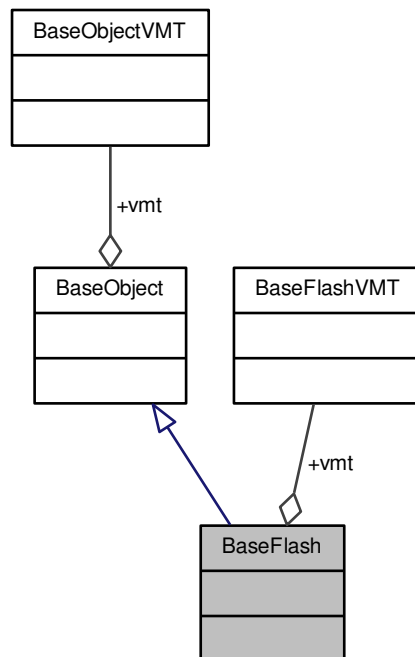
Base flash class.

```
#include <hal_flash.h>
```


Inheritance diagram for BaseFlash:



Collaboration diagram for BaseFlash:



Data Fields

- const struct [BaseFlashVMT](#) * [vmt](#)
Virtual Methods Table.

8.16.1 Detailed Description

Base flash class.

8.16.2 Field Documentation

8.16.2.1 const struct **BaseFlashVMT*** **BaseFlash::vmt**

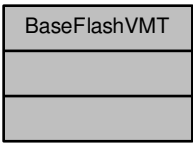
Virtual Methods Table.

8.17 BaseFlashVMT Struct Reference

[BaseFlash](#) virtual methods table.

```
#include <hal_flash.h>
```

Collaboration diagram for BaseFlashVMT:



8.17.1 Detailed Description

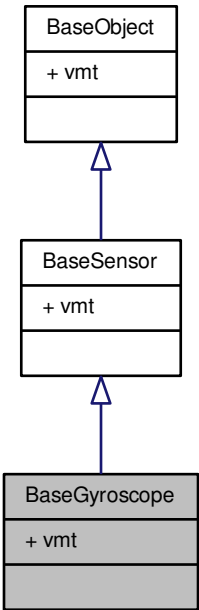
BaseFlash virtual methods table.

8.18 BaseGyroscope Struct Reference

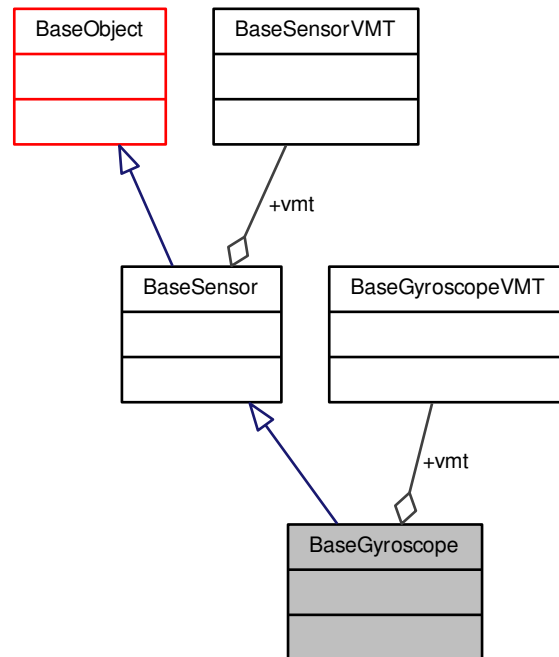
Base gyroscope class.

```
#include <hal_gyroscope.h>
```

Inheritance diagram for BaseGyroscope:



Collaboration diagram for BaseGyroscope:



Data Fields

- const struct [BaseGyroscopeVMT](#) * vmt

Virtual Methods Table.

8.18.1 Detailed Description

Base gyroscope class.

This class represents a generic gyroscope.

8.18.2 Field Documentation

8.18.2.1 const struct [BaseGyroscopeVMT](#)* BaseGyroscope::vmt

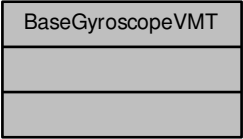
Virtual Methods Table.

8.19 BaseGyroscopeVMT Struct Reference

[BaseGyroscope](#) virtual methods table.

```
#include <hal_gyroscope.h>
```

Collaboration diagram for BaseGyroscopeVMT:



8.19.1 Detailed Description

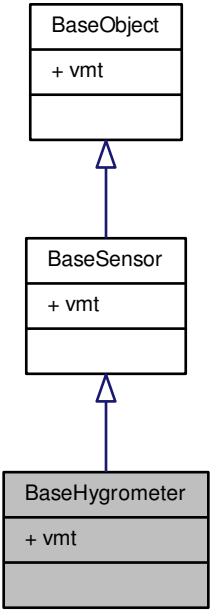
BaseGyroscope virtual methods table.

8.20 BaseHygrometer Struct Reference

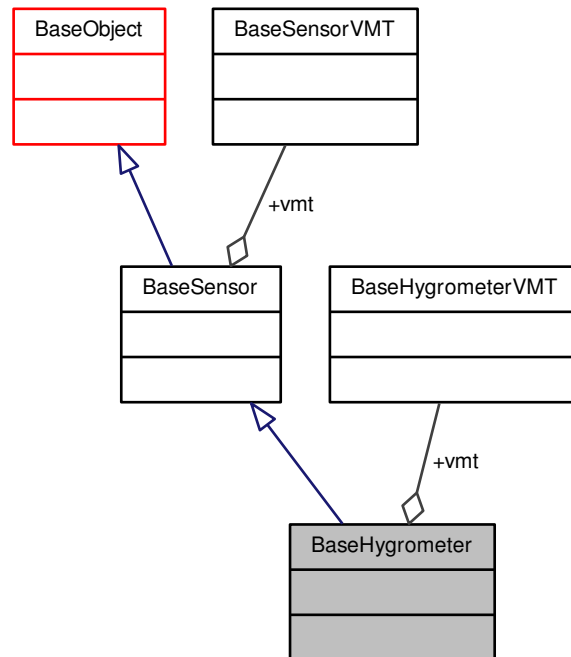
Base hygrometer class.

```
#include <hal_hygrometer.h>
```

Inheritance diagram for BaseHygrometer:



Collaboration diagram for BaseHygrometer:



Data Fields

- const struct [BaseHygrometerVMT](#) * **vmt**

Virtual Methods Table.

8.20.1 Detailed Description

Base hygrometer class.

This class represents a generic hygrometer.

8.20.2 Field Documentation

8.20.2.1 const struct **BaseHygrometerVMT*** **BaseHygrometer::vmt**

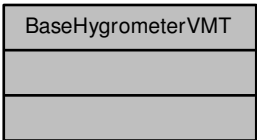
Virtual Methods Table.

8.21 BaseHygrometerVMT Struct Reference

[BaseHygrometer](#) virtual methods table.

```
#include <hal_hygrometer.h>
```

Collaboration diagram for BaseHygrometerVMT:



8.21.1 Detailed Description

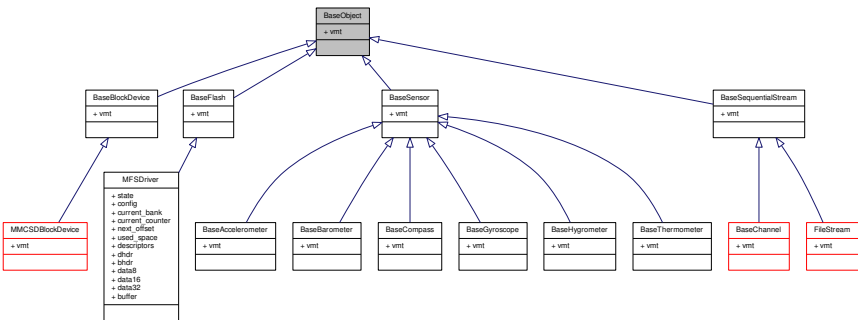
BaseHygrometer virtual methods table.

8.22 BaseObject Struct Reference

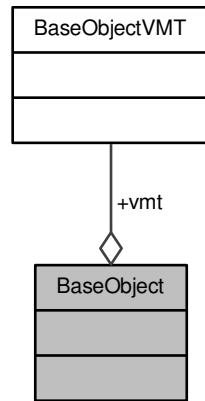
Base stream class.

```
#include <hal_objects.h>
```

Inheritance diagram for BaseObject:



Collaboration diagram for BaseObject:



Data Fields

- const struct [BaseObjectVMT](#) * [vmt](#)

Virtual Methods Table.

8.22.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

8.22.2 Field Documentation

8.22.2.1 const struct **BaseObjectVMT*** **BaseObject::vmt**

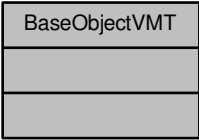
Virtual Methods Table.

8.23 BaseObjectVMT Struct Reference

[BaseObject](#) virtual methods table.

```
#include <hal_objects.h>
```


Collaboration diagram for BaseObjectVMT:



8.23.1 Detailed Description

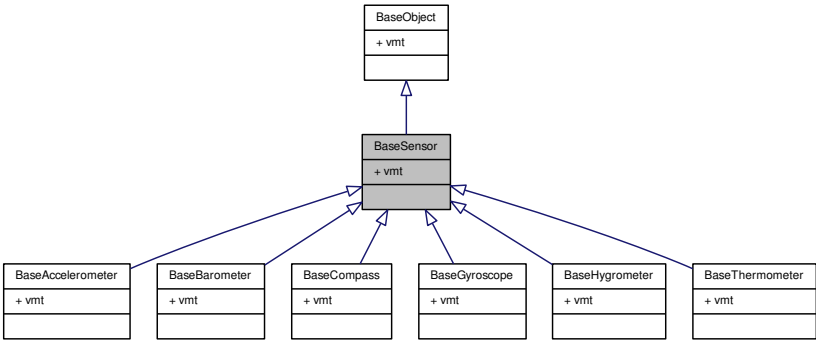
BaseObject virtual methods table.

8.24 BaseSensor Struct Reference

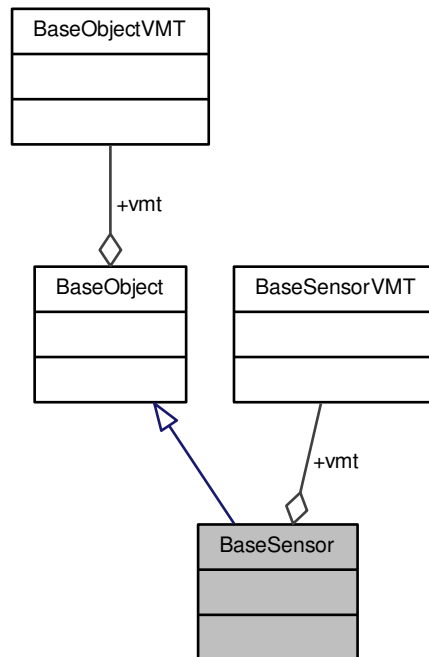
Base stream class.

```
#include <hal_sensors.h>
```

Inheritance diagram for BaseSensor:



Collaboration diagram for BaseSensor:



Data Fields

- const struct [BaseSensorVMT](#) * [vmt](#)
Virtual Methods Table.

8.24.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

8.24.2 Field Documentation

8.24.2.1 const struct [BaseSensorVMT](#)* [BaseSensor::vmt](#)

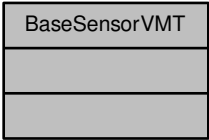
Virtual Methods Table.

8.25 BaseSensorVMT Struct Reference

[BaseSensor](#) virtual methods table.

```
#include <hal_sensors.h>
```

Collaboration diagram for BaseSensorVMT:



8.25.1 Detailed Description

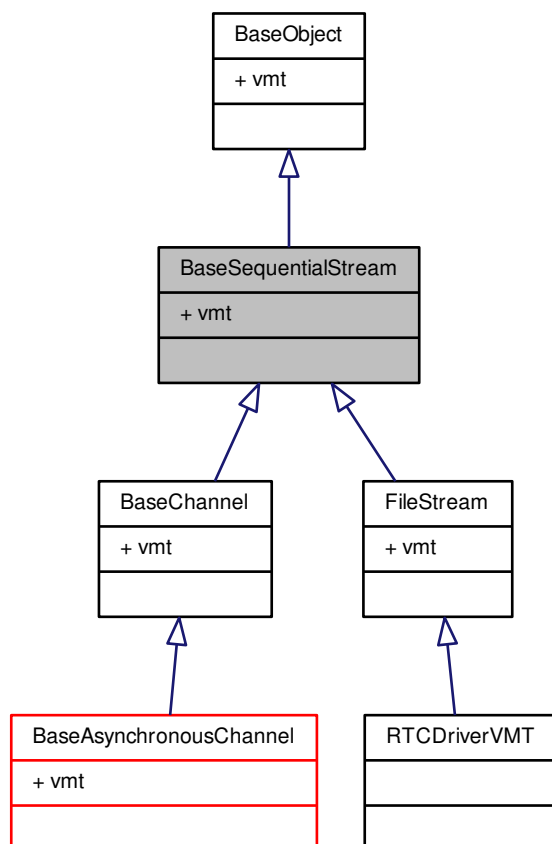
BaseSensor virtual methods table.

8.26 BaseSequentialStream Struct Reference

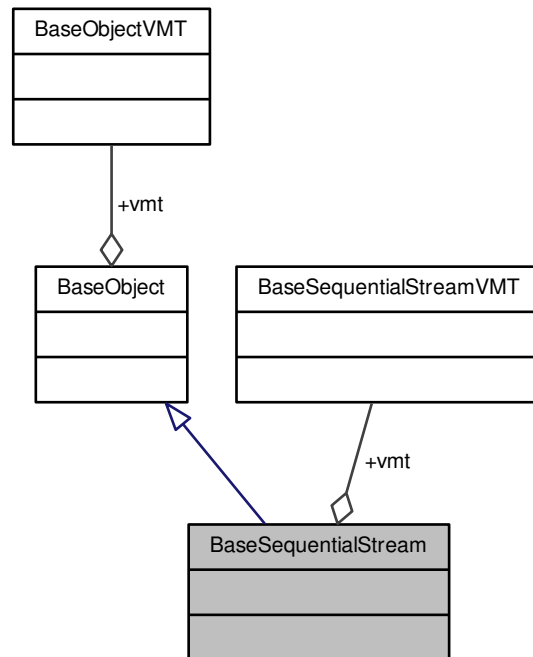
Base stream class.

```
#include <hal_streams.h>
```

Inheritance diagram for BaseSequentialStream:



Collaboration diagram for BaseSequentialStream:



Data Fields

- const struct [BaseSequentialStreamVMT](#) * vmt

Virtual Methods Table.

8.26.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

8.26.2 Field Documentation

8.26.2.1 const struct [BaseSequentialStreamVMT](#)* BaseSequentialStream::vmt

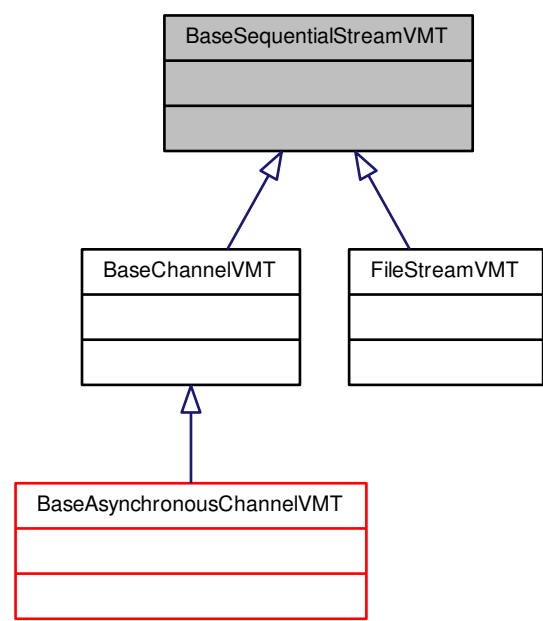
Virtual Methods Table.

8.27 BaseSequentialStreamVMT Struct Reference

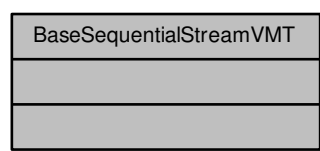
[BaseSequentialStream](#) virtual methods table.

```
#include <hal_streams.h>
```

Inheritance diagram for BaseSequentialStreamVMT:



Collaboration diagram for BaseSequentialStreamVMT:



8.27.1 Detailed Description

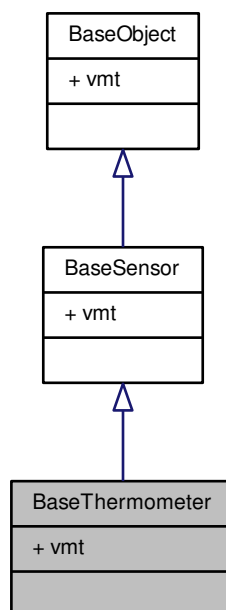
`BaseSequentialStream` virtual methods table.

8.28 BaseThermometer Struct Reference

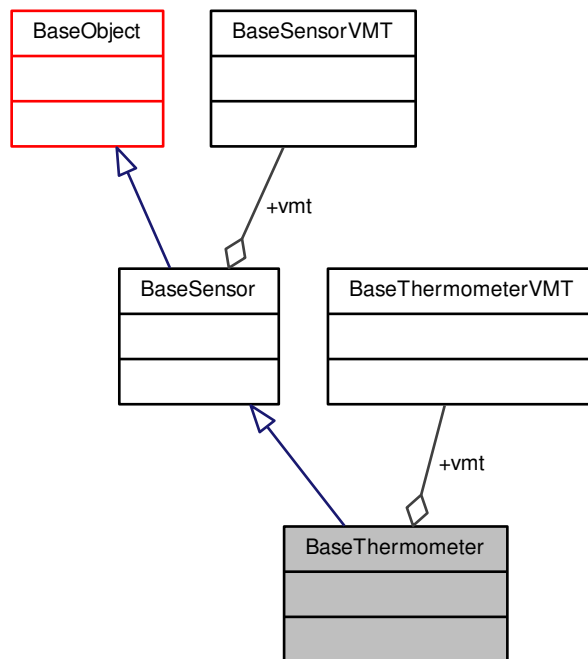
Base thermometer class.

```
#include <hal_thermometer.h>
```

Inheritance diagram for BaseThermometer:



Collaboration diagram for BaseThermometer:



Data Fields

- const struct [BaseThermometerVMT](#) * vmt

Virtual Methods Table.

8.28.1 Detailed Description

Base thermometer class.

This class represents a generic thermometer.

8.28.2 Field Documentation

8.28.2.1 const struct [BaseThermometerVMT](#)* BaseThermometer::vmt

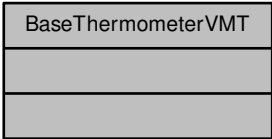
Virtual Methods Table.

8.29 BaseThermometerVMT Struct Reference

[BaseThermometer](#) virtual methods table.

```
#include <hal_thermometer.h>
```


Collaboration diagram for BaseThermometerVMT:



8.29.1 Detailed Description

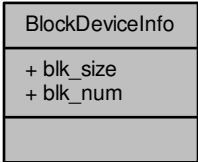
BaseThermometer virtual methods table.

8.30 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.

8.30.1 Detailed Description

Block device info.

8.30.2 Field Documentation

8.30.2.1 `uint32_t BlockDeviceInfo::blk_size`

Block size in bytes.

8.30.2.2 `uint32_t BlockDeviceInfo::blk_num`

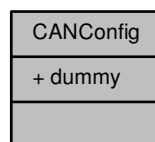
Total number of blocks.

8.31 CANConfig Struct Reference

Driver configuration structure.

```
#include <hal_can_lld.h>
```

Collaboration diagram for CANConfig:



8.31.1 Detailed Description

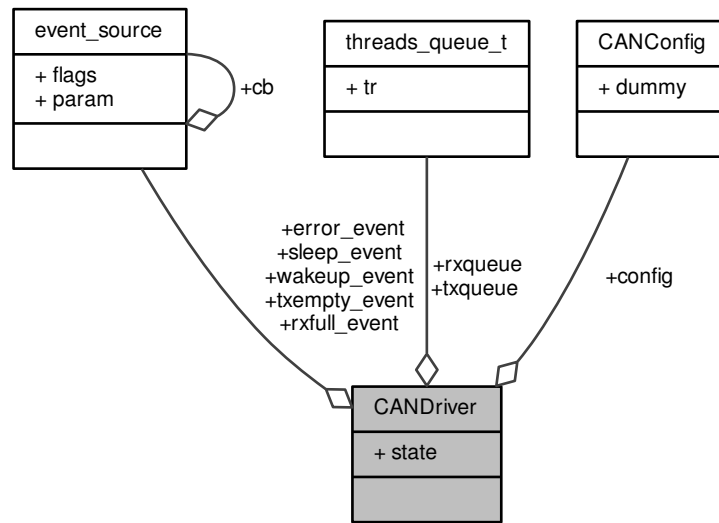
Driver configuration structure.

8.32 CANDriver Struct Reference

Structure representing an CAN driver.

```
#include <hal_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.

8.32.1 Detailed Description

Structure representing an CAN driver.

8.32.2 Field Documentation

8.32.2.1 `canstate_t` `CANDriver::state`

Driver state.

8.32.2.2 `const CANConfig*` `CANDriver::config`

Current configuration data.

8.32.2.3 `threads_queue_t` `CANDriver::txqueue`

Transmission threads queue.

8.32.2.4 `threads_queue_t` `CANDriver::rxqueue`

Receive threads queue.

8.32.2.5 `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.

8.32.2.6 `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.

8.32.2.7 `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.

8.32.2.8 `event_source_t` `CANDriver::sleep_event`

Entering sleep state event.

8.32.2.9 `event_source_t` `CANDriver::wakeup_event`

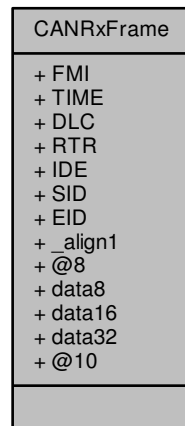
Exiting sleep state event.

8.33 CANRxFrame Struct Reference

CAN received frame.

```
#include <hal_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.

8.33.1 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.

8.33.2 Field Documentation

8.33.2.1 `uint8_t CANRxFrame::FMI`

Filter id.

8.33.2.2 `uint16_t CANRxFrame::TIME`

Time stamp.

8.33.2.3 `uint8_t CANRxFrame::DLC`

Data length.

8.33.2.4 `uint8_t CANRxFrame::RTR`

Frame type.

8.33.2.5 `uint8_t CANRxFrame::IDE`

Identifier type.

8.33.2.6 `uint32_t CANRxFrame::SID`

Standard identifier.

8.33.2.7 `uint32_t CANRxFrame::EID`

Extended identifier.

8.33.2.8 `uint8_t CANRxFrame::data8[8]`

Frame data.

8.33.2.9 `uint16_t CANRxFrame::data16[4]`

Frame data.

8.33.2.10 `uint32_t CANRxFrame::data32[2]`

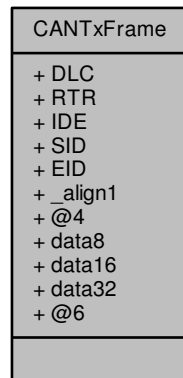
Frame data.

8.34 CANTxFrame Struct Reference

CAN transmission frame.

```
#include <hal_can_lld.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.

8.34.1 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.

8.34.2 Field Documentation

8.34.2.1 `uint8_t CANTxFrame::DLC`

Data length.

8.34.2.2 `uint8_t CANTxFrame::RTR`

Frame type.

8.34.2.3 `uint8_t CANTxFrame::IDE`

Identifier type.

8.34.2.4 `uint32_t CANTxFrame::SID`

Standard identifier.

8.34.2.5 `uint32_t CANTxFrame::EID`

Extended identifier.

8.34.2.6 `uint8_t CANTxFrame::data8[8]`

Frame data.

8.34.2.7 `uint16_t CANTxFrame::data16[4]`

Frame data.

8.34.2.8 `uint32_t CANTxFrame::data32[2]`

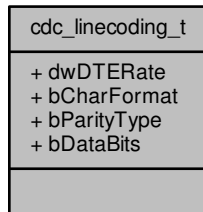
Frame data.

8.35 `cdc_linecoding_t` Struct Reference

Type of Line Coding structure.

```
#include <hal_usb_cdc.h>
```


Collaboration diagram for cdc_linecoding_t:



8.35.1 Detailed Description

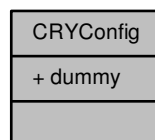
Type of Line Coding structure.

8.36 CRYConfig Struct Reference

Driver configuration structure.

```
#include <hal_crypto_llid.h>
```

Collaboration diagram for CRYConfig:



8.36.1 Detailed Description

Driver configuration structure.

Note

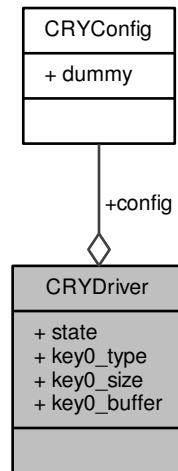
It could be empty on some architectures.

8.37 CRYDriver Struct Reference

Structure representing an CRY driver.

```
#include <hal_crypto_llid.h>
```

Collaboration diagram for CRYDriver:



Data Fields

- [crystate_t state](#)
Driver state.
- `const CRYConfig * config`
Current configuration data.
- [cryalgorithm_t key0_type](#)
Algorithm type of transient key.
- `size_t key0_size`
Size of transient key.
- `uint8_t key0_buffer [HAL_CRY_MAX_KEY_SIZE]`
Key buffer for the fall-back implementation.

8.37.1 Detailed Description

Structure representing an CRY driver.

8.37.2 Field Documentation

8.37.2.1 `crystate_t CRYDriver::state`

Driver state.

8.37.2.2 `const CRYConfig* CRYDriver::config`

Current configuration data.

8.37.2.3 `cryalgorithm_t` CRYDriver::key0_type

Algorithm type of transient key.

8.37.2.4 `size_t` CRYDriver::key0_size

Size of transient key.

8.37.2.5 `uint8_t` CRYDriver::key0_buffer[HAL_CRY_MAX_KEY_SIZE]

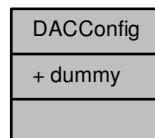
Key buffer for the fall-back implementation.

8.38 DACConfig Struct Reference

Driver configuration structure.

```
#include <hal_dac_llld.h>
```

Collaboration diagram for DACConfig:



8.38.1 Detailed Description

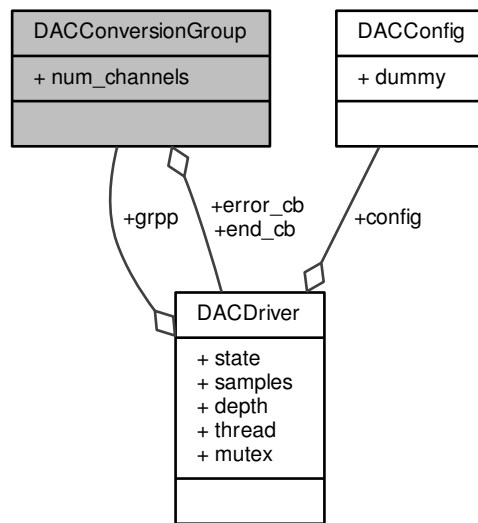
Driver configuration structure.

8.39 DACConversionGroup Struct Reference

DAC Conversion group structure.

```
#include <hal_dac_llld.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`.

8.39.1 Detailed Description

DAC Conversion group structure.

8.39.2 Field Documentation

8.39.2.1 `uint32_t DACConversionGroup::num_channels`

Number of DAC channels.

8.39.2.2 `daccallback_t DACConversionGroup::end_cb`

Operation complete callback or `NULL`.

8.39.2.3 `dacerrorcallback_t DACConversionGroup::error_cb`

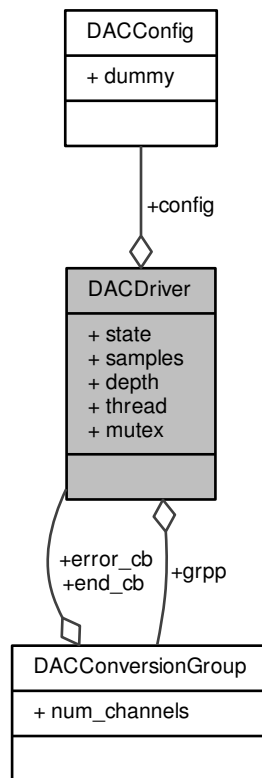
Error handling callback or `NULL`.

8.40 DACDriver Struct Reference

Structure representing a DAC driver.

```
#include <hal_dac_llh.h>
```

Collaboration diagram for DACDriver:



Data Fields

- [dacstate_t state](#)
Driver state.
- [const DACConversionGroup * grpp](#)
Conversion group.
- [dacsamples_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.

8.40.1 Detailed Description

Structure representing a DAC driver.

8.40.2 Field Documentation

8.40.2.1 `dacstate_t DACDriver::state`

Driver state.

8.40.2.2 `const DACConversionGroup* DACDriver::grpp`

Conversion group.

8.40.2.3 `dacsample_t* DACDriver::samples`

Samples buffer pointer.

8.40.2.4 `uint16_t DACDriver::depth`

Samples buffer size.

8.40.2.5 `const DACConfig* DACDriver::config`

Current configuration data.

8.40.2.6 `thread_reference_t DACDriver::thread`

Waiting thread.

8.40.2.7 `mutex_t DACDriver::mutex`

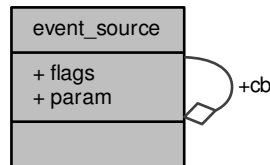
Mutex protecting the bus.

8.41 `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.

8.41.1 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.

8.41.2 Field Documentation

8.41.2.1 volatile [eventflags_t](#) [event_source::flags](#)

Stored event flags.

8.41.2.2 [eventcallback_t](#) [event_source::cb](#)

Event source callback.

8.41.2.3 void* [event_source::param](#)

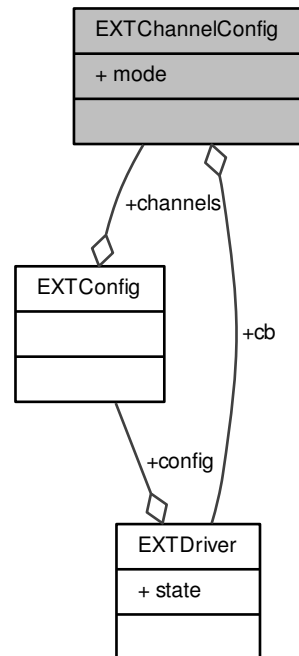
User defined field.

8.42 EXTChannelConfig Struct Reference

Channel configuration structure.

```
#include <hal_ext_lld.h>
```

Collaboration diagram for EXTChannelConfig:



Data Fields

- `uint32_t mode`
Channel mode.
- `extcallback_t cb`
Channel callback.

8.42.1 Detailed Description

Channel configuration structure.

8.42.2 Field Documentation

8.42.2.1 `uint32_t EXTChannelConfig::mode`

Channel mode.

8.42.2.2 `extcallback_t EXTChannelConfig::cb`

Channel callback.

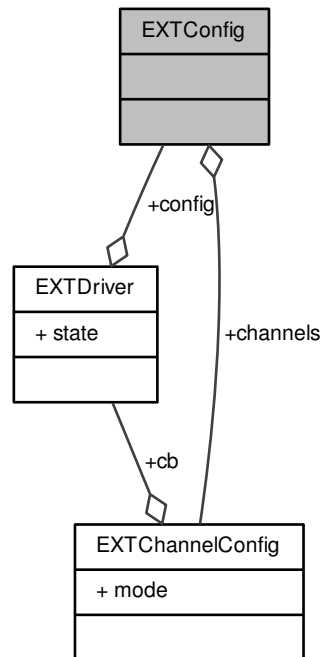
In the STM32 implementation a `NULL` callback pointer is valid and configures the channel as an event sources instead of an interrupt source.

8.43 EXTConfig Struct Reference

Driver configuration structure.

```
#include <hal_ext_lld.h>
```

Collaboration diagram for EXTConfig:



Data Fields

- [EXTChannelConfig channels \[EXT_MAX_CHANNELS\]](#)
Channel configurations.

8.43.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.43.2 Field Documentation

8.43.2.1 EXTChannelConfig EXTConfig::channels[EXT_MAX_CHANNELS]

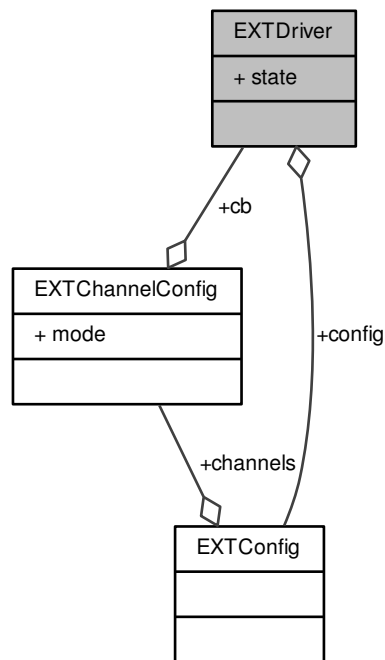
Channel configurations.

8.44 EXTDriver Struct Reference

Structure representing an EXT driver.

```
#include <hal_ext_lld.h>
```

Collaboration diagram for EXTDriver:



Data Fields

- [extstate_t state](#)
Driver state.
- `const EXTConfig * config`
Current configuration data.

8.44.1 Detailed Description

Structure representing an EXT driver.

8.44.2 Field Documentation

8.44.2.1 `extstate_t EXTDriver::state`

Driver state.

8.44.2.2 `const EXTConfig* EXTDriver::config`

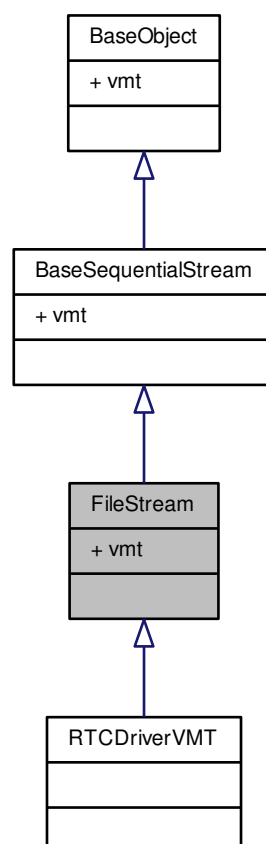
Current configuration data.

8.45 FileStream Struct Reference

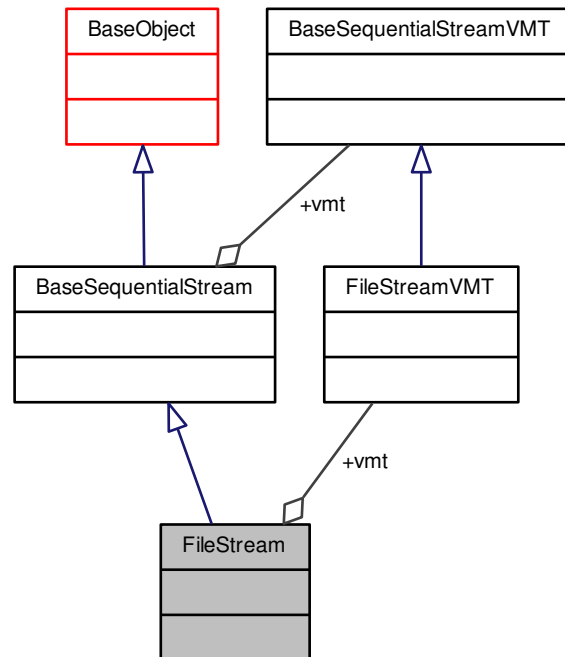
Base file stream class.

```
#include <hal_files.h>
```

Inheritance diagram for FileStream:



Collaboration diagram for FileStream:



Data Fields

- const struct [FileStreamVMT](#) * [vmt](#)

Virtual Methods Table.

8.45.1 Detailed Description

Base file stream class.

This class represents a generic file data stream.

8.45.2 Field Documentation

8.45.2.1 const struct [FileStreamVMT](#)* [FileStream::vmt](#)

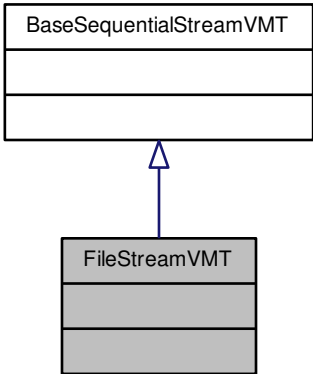
Virtual Methods Table.

8.46 FileStreamVMT Struct Reference

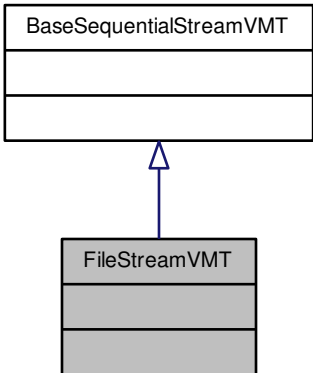
[FileStream](#) virtual methods table.

```
#include <hal_files.h>
```

Inheritance diagram for FileStreamVMT:



Collaboration diagram for FileStreamVMT:



8.46.1 Detailed Description

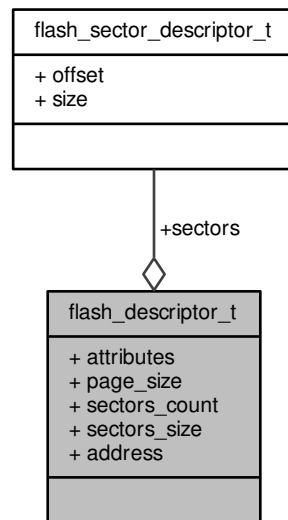
`FileStream` virtual methods table.

8.47 flash_descriptor_t Struct Reference

Type of a flash device descriptor.

```
#include <hal_flash.h>
```

Collaboration diagram for `flash_descriptor_t`:



Data Fields

- `uint32_t` [attributes](#)
Device_attributes.
- `uint32_t` [page_size](#)
Size of write page.
- `flash_sector_t` [sectors_count](#)
Number of sectors in the device.
- `const flash_sector_descriptor_t *` [sectors](#)
List of sectors for devices with non-uniform sector sizes.
- `uint32_t` [sectors_size](#)
Size of sectors for devices with uniform sector size.
- `flash_offset_t` [address](#)
Flash address if memory mapped or zero.

8.47.1 Detailed Description

Type of a flash device descriptor.

8.47.2 Field Documentation

8.47.2.1 `uint32_t flash_descriptor_t::attributes`

Device_attributes.

8.47.2.2 uint32_t flash_descriptor_t::page_size

Size of write page.

8.47.2.3 flash_sector_t flash_descriptor_t::sectors_count

Number of sectors in the device.

8.47.2.4 const flash_sector_descriptor_t* flash_descriptor_t::sectors

List of sectors for devices with non-uniform sector sizes.

Note

If `NULL` then the device has uniform sectors size equal to `sector_size`.

8.47.2.5 uint32_t flash_descriptor_t::sectors_size

Size of sectors for devices with uniform sector size.

Note

If zero then the device has non uniform sectors described by the `sectors` array.

8.47.2.6 flash_offset_t flash_descriptor_t::address

Flash address if memory mapped or zero.

Note

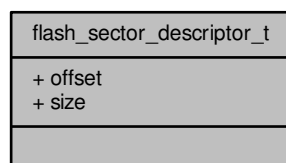
Conventionally, non memory mapped devices have address zero.

8.48 flash_sector_descriptor_t Struct Reference

Flash sector descriptor.

```
#include <hal_flash.h>
```

Collaboration diagram for `flash_sector_descriptor_t`:



Data Fields

- [flash_offset_t offset](#)
Sector offset.
- [uint32_t size](#)
Sector size.

8.48.1 Detailed Description

Flash sector descriptor.

8.48.2 Field Documentation

8.48.2.1 [flash_offset_t flash_sector_descriptor_t::offset](#)

Sector offset.

8.48.2.2 [uint32_t flash_sector_descriptor_t::size](#)

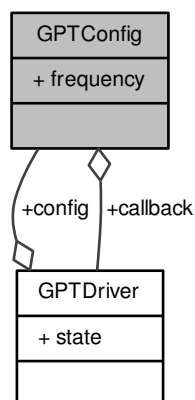
Sector size.

8.49 GPTConfig Struct Reference

Driver configuration structure.

```
#include <hal_gpt_lld.h>
```

Collaboration diagram for GPTConfig:



Data Fields

- [gptfreq_t frequency](#)

Timer clock in Hz.

- [gptcallback_t callback](#)

Timer callback pointer.

8.49.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.49.2 Field Documentation

8.49.2.1 gptfreq_t GPTConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.49.2.2 gptcallback_t GPTConfig::callback

Timer callback pointer.

Note

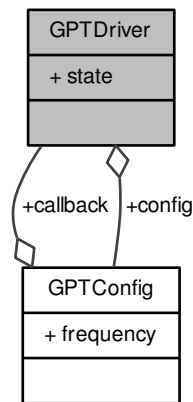
This callback is invoked on GPT counter events.

8.50 GPTDriver Struct Reference

Structure representing a GPT driver.

```
#include <hal_gpt_lld.h>
```

Collaboration diagram for GPTDriver:



Data Fields

- [gptstate_t state](#)
Driver state.
- [const GPTConfig * config](#)
Current configuration data.

8.50.1 Detailed Description

Structure representing a GPT driver.

8.50.2 Field Documentation

8.50.2.1 `gptstate_t GPTDriver::state`

Driver state.

8.50.2.2 `const GPTConfig* GPTDriver::config`

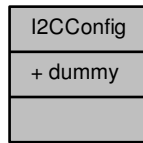
Current configuration data.

8.51 I2CConfig Struct Reference

Type of I2C driver configuration structure.

```
#include <hal_i2c_llc.h>
```

Collaboration diagram for I2CConfig:



8.51.1 Detailed Description

Type of I2C driver configuration structure.

Note

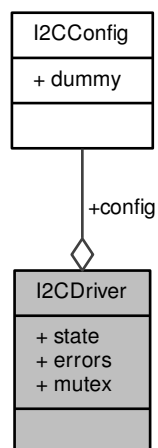
Implementations may extend this structure to contain more, architecture dependent, fields.

8.52 I2CDriver Struct Reference

Structure representing an I2C driver.

```
#include <hal_i2c_llid.h>
```

Collaboration diagram for I2CDriver:



Data Fields

- [i2cstate_t state](#)

Driver state.

- `const I2CConfig * config`

Current configuration data.

- `i2cflags_t errors`

Error flags.

8.52.1 Detailed Description

Structure representing an I2C driver.

8.52.2 Field Documentation

8.52.2.1 `i2cstate_t I2CDriver::state`

Driver state.

8.52.2.2 `const I2CConfig* I2CDriver::config`

Current configuration data.

8.52.2.3 `i2cflags_t I2CDriver::errors`

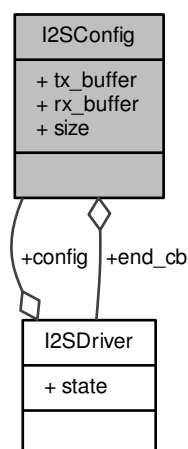
Error flags.

8.53 I2SConfig Struct Reference

Driver configuration structure.

```
#include <hal_i2s_lld.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.

8.53.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.53.2 Field Documentation

8.53.2.1 `const void* I2SConfig::tx_buffer`

Transmission buffer pointer.

Note

Can be `NULL` if TX is not required.

8.53.2.2 `void* I2SConfig::rx_buffer`

Receive buffer pointer.

Note

Can be `NULL` if RX is not required.

8.53.2.3 `size_t I2SConfig::size`

TX and RX buffers size as number of samples.

8.53.2.4 `i2scallback_t I2SConfig::end_cb`

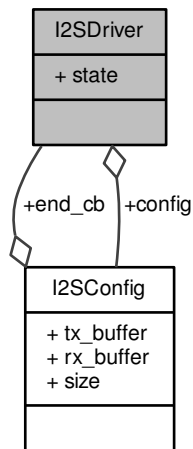
Callback function called during streaming.

8.54 I2SDriver Struct Reference

Structure representing an I2S driver.

```
#include <hal_i2s_ll.h>
```

Collaboration diagram for I2SDriver:



Data Fields

- [i2sstate_t state](#)
Driver state.
- `const I2SConfig * config`
Current configuration data.

8.54.1 Detailed Description

Structure representing an I2S driver.

8.54.2 Field Documentation

8.54.2.1 `i2sstate_t I2SDriver::state`

Driver state.

8.54.2.2 `const I2SConfig* I2SDriver::config`

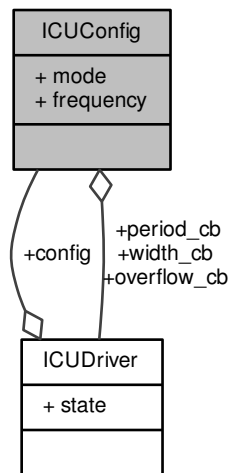
Current configuration data.

8.55 ICUConfig Struct Reference

Driver configuration structure.

```
#include <hal_icu_llld.h>
```

Collaboration diagram for ICUConfig:



Data Fields

- [icumode_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.

8.55.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.55.2 Field Documentation

8.55.2.1 [icumode_t ICUConfig::mode](#)

Driver mode.

8.55.2.2 `icufreq_t` `ICUConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.55.2.3 `icucallback_t` `ICUConfig::width_cb`

Callback for pulse width measurement.

8.55.2.4 `icucallback_t` `ICUConfig::period_cb`

Callback for cycle period measurement.

8.55.2.5 `icucallback_t` `ICUConfig::overflow_cb`

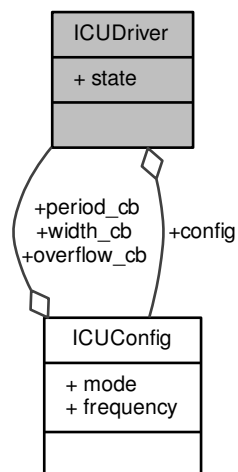
Callback for timer overflow.

8.56 ICUDriver Struct Reference

Structure representing an ICU driver.

```
#include <hal_icu_lld.h>
```

Collaboration diagram for ICUDriver:



Data Fields

- [icustate_t state](#)

Driver state.

- const ICUConfig * config

Current configuration data.

8.56.1 Detailed Description

Structure representing an ICU driver.

8.56.2 Field Documentation

8.56.2.1 icustate_t ICUDriver::state

Driver state.

8.56.2.2 const ICUConfig* ICUDriver::config

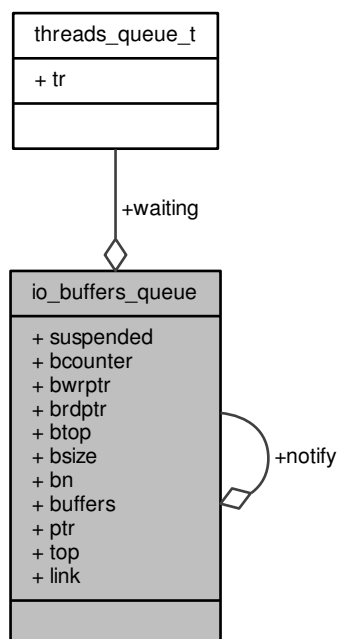
Current configuration data.

8.57 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.
- `bool` [suspended](#)
Queue suspended state flag.
- `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.

8.57.1 Detailed Description

Structure of a generic buffers queue.

8.57.2 Field Documentation

8.57.2.1 `threads_queue_t` `io_buffers_queue::waiting`

Queue of waiting threads.

8.57.2.2 `bool` `io_buffers_queue::suspended`

Queue suspended state flag.

8.57.2.3 `volatile size_t` `io_buffers_queue::bcounter`

Active buffers counter.

8.57.2.4 `uint8_t*` `io_buffers_queue::bwrptr`

Buffer write pointer.

8.57.2.5 `uint8_t* io_buffers_queue::brdptr`

Buffer read pointer.

8.57.2.6 `uint8_t* io_buffers_queue::btop`

Pointer to the buffers boundary.

8.57.2.7 `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.

8.57.2.8 `size_t io_buffers_queue::bn`

Number of buffers.

8.57.2.9 `uint8_t* io_buffers_queue::buffers`

Queue of buffer objects.

8.57.2.10 `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.

8.57.2.11 `uint8_t* io_buffers_queue::top`

Boundary for R/W sequential access.

8.57.2.12 `bqnotify_t io_buffers_queue::notify`

Data notification callback.

8.57.2.13 `void* io_buffers_queue::link`

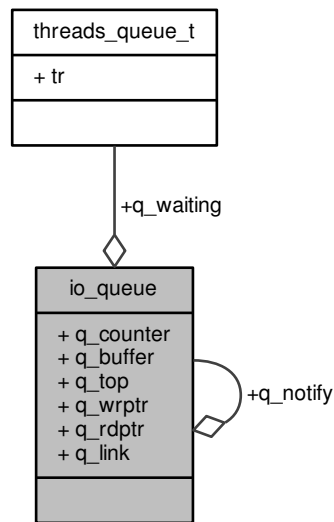
Application defined field.

8.58 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](#)
Pointer to the first location after the buffer.
- uint8_t * [q_wrptr](#)
Write pointer.
- uint8_t * [q_rdptra](#)
Read pointer.
- [qnotify_t q_notify](#)
Data notification callback.
- void * [q_link](#)
Application defined field.

8.58.1 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.

8.58.2 Field Documentation

8.58.2.1 `threads_queue_t io_queue::q_waiting`

Queue of waiting threads.

8.58.2.2 `volatile size_t io_queue::q_counter`

Resources counter.

8.58.2.3 `uint8_t* io_queue::q_buffer`

Pointer to the queue buffer.

8.58.2.4 `uint8_t* io_queue::q_top`

Pointer to the first location after the buffer.

8.58.2.5 `uint8_t* io_queue::q_wrptr`

Write pointer.

8.58.2.6 `uint8_t* io_queue::q_rdptr`

Read pointer.

8.58.2.7 `qnotify_t io_queue::q_notify`

Data notification callback.

8.58.2.8 `void* io_queue::q_link`

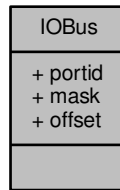
Application defined field.

8.59 IOBus Struct Reference

I/O bus descriptor.

```
#include <hal_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.

8.59.1 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.

8.59.2 Field Documentation

8.59.2.1 [ioportid_t IOBus::portid](#)

Port identifier.

8.59.2.2 [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.

8.59.2.3 [uint_fast8_t IOBus::offset](#)

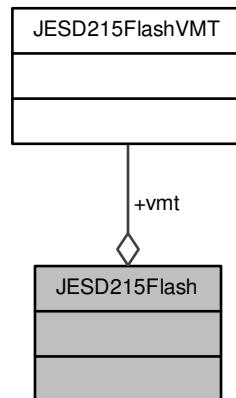
Offset, within the port, of the least significant bit of the bus.

8.60 JESD215Flash Struct Reference

Base flash class.

```
#include <hal_jesd216_flash.h>
```

Collaboration diagram for JESD215Flash:



Data Fields

- const struct [JESD215FlashVMT](#) * vmt

Virtual Methods Table.

8.60.1 Detailed Description

Base flash class.

8.60.2 Field Documentation

8.60.2.1 const struct [JESD215FlashVMT](#)* JESD215Flash::vmt

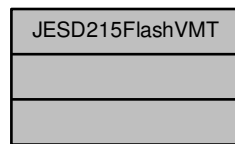
Virtual Methods Table.

8.61 JESD215FlashVMT Struct Reference

[JESD215Flash](#) virtual methods table.

```
#include <hal_jesd216_flash.h>
```

Collaboration diagram for JESD215FlashVMT:



8.61.1 Detailed Description

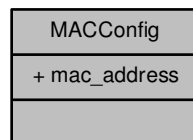
[JESD215Flash](#) virtual methods table.

8.62 MACConfig Struct Reference

Driver configuration structure.

```
#include <hal_mac_llld.h>
```

Collaboration diagram for MACConfig:



Data Fields

- `uint8_t * mac_address`
MAC address.

8.62.1 Detailed Description

Driver configuration structure.

8.62.2 Field Documentation

8.62.2.1 `uint8_t* MACConfig::mac_address`

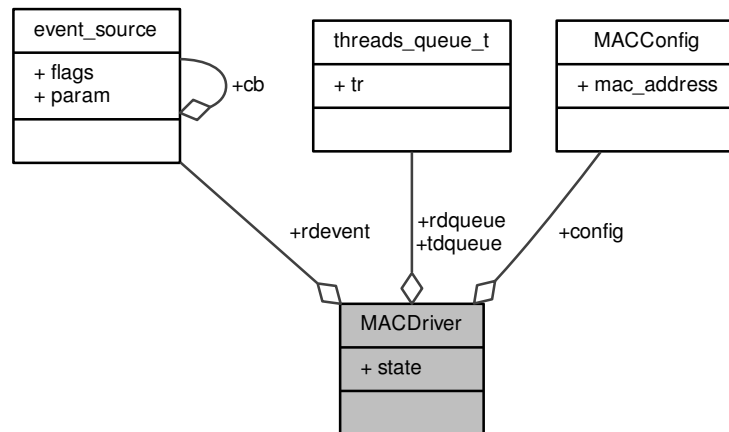
MAC address.

8.63 MACDriver Struct Reference

Structure representing a MAC driver.

```
#include <hal_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.

8.63.1 Detailed Description

Structure representing a MAC driver.

8.63.2 Field Documentation

8.63.2.1 `macstate_t MACDriver::state`

Driver state.

8.63.2.2 `const MACConfig* MACDriver::config`

Current configuration data.

8.63.2.3 `threads_queue_t MACDriver::tdqueue`

Transmit semaphore.

8.63.2.4 `threads_queue_t MACDriver::rdqueue`

Receive semaphore.

8.63.2.5 `event_source_t MACDriver::rdevent`

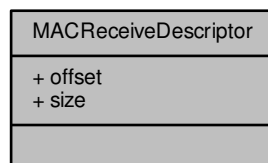
Receive event.

8.64 MACReceiveDescriptor Struct Reference

Structure representing a receive descriptor.

```
#include <hal_mac_lld.h>
```

Collaboration diagram for MACReceiveDescriptor:



Data Fields

- `size_t offset`
Current read offset.
- `size_t size`
Available data size.

8.64.1 Detailed Description

Structure representing a receive descriptor.

8.64.2 Field Documentation

8.64.2.1 `size_t MACReceiveDescriptor::offset`

Current read offset.

8.64.2.2 `size_t MACReceiveDescriptor::size`

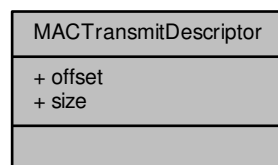
Available data size.

8.65 MACTransmitDescriptor Struct Reference

Structure representing a transmit descriptor.

```
#include <hal_mac_lld.h>
```

Collaboration diagram for MACTransmitDescriptor:



Data Fields

- `size_t offset`
Current write offset.
- `size_t size`
Available space size.

8.65.1 Detailed Description

Structure representing a transmit descriptor.

8.65.2 Field Documentation

8.65.2.1 `size_t MACTransmitDescriptor::offset`

Current write offset.

8.65.2.2 `size_t MACTransmitDescriptor::size`

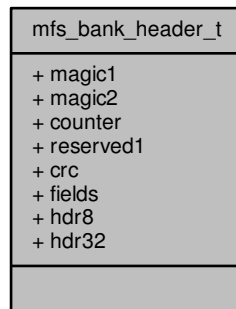
Available space size.

8.66 mfs_bank_header_t Union Reference

Type of a bank header.

```
#include <mfs.h>
```

Collaboration diagram for mfs_bank_header_t:



8.66.1 Detailed Description

Type of a bank header.

Note

The header resides in the first 16 bytes of a bank.

8.66.2 Field Documentation

8.66.2.1 `uint32_t mfs_bank_header_t::magic1`

Bank magic 1.

8.66.2.2 `uint32_t mfs_bank_header_t::magic2`

Bank magic 2.

8.66.2.3 `uint32_t mfs_bank_header_t::counter`

Usage counter of the bank.

This value is increased each time a bank swap is performed. It indicates how much wearing the flash has already endured.

8.66.2.4 `uint16_t mfs_bank_header_t::reserved1`

Reserved field.

8.66.2.5 uint16_t mfs_bank_header_t::crc

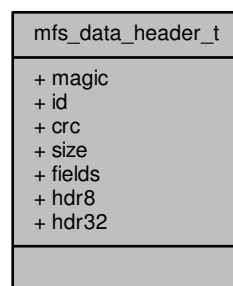
Header CRC.

8.67 mfs_data_header_t Union Reference

Type of a data block header.

```
#include <mfs.h>
```

Collaboration diagram for mfs_data_header_t:



8.67.1 Detailed Description

Type of a data block header.

This structure is placed before each written data block.

8.67.2 Field Documentation

8.67.2.1 uint32_t mfs_data_header_t::magic

Data header magic.

8.67.2.2 uint16_t mfs_data_header_t::id

Data identifier.

8.67.2.3 uint16_t mfs_data_header_t::crc

Data CRC.

8.67.2.4 uint32_t mfs_data_header_t::size

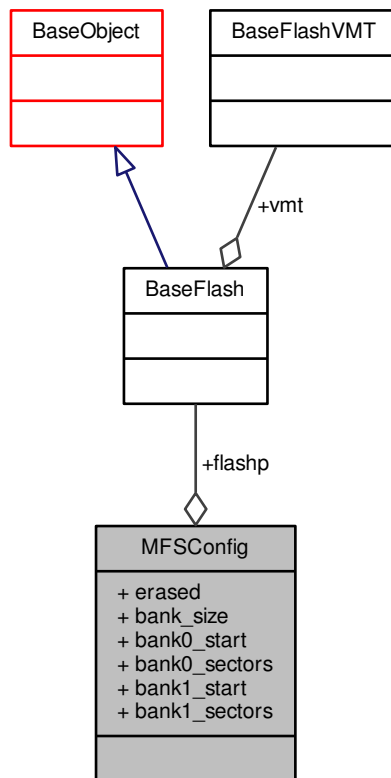
Data size.

8.68 MFSConfig Struct Reference

Type of a MFS configuration structure.

```
#include <mfs.h>
```

Collaboration diagram for MFSConfig:



Data Fields

- [BaseFlash * flashp](#)
Flash driver associated to this MFS instance.
- [uint32_t erased](#)
Erased value.
- [flash_offset_t bank_size](#)
Banks size.
- [flash_sector_t bank0_start](#)
Base sector index for bank 0.
- [flash_sector_t bank0_sectors](#)
Number of sectors for bank 0.
- [flash_sector_t bank1_start](#)
Base sector index for bank 1.
- [flash_sector_t bank1_sectors](#)
Number of sectors for bank 1.

8.68.1 Detailed Description

Type of a MFS configuration structure.

8.68.2 Field Documentation

8.68.2.1 BaseFlash* MFSCfg::flashp

Flash driver associated to this MFS instance.

8.68.2.2 uint32_t MFSCfg::erased

Erased value.

8.68.2.3 flash_offset_t MFSCfg::bank_size

Banks size.

8.68.2.4 flash_sector_t MFSCfg::bank0_start

Base sector index for bank 0.

8.68.2.5 flash_sector_t MFSCfg::bank0_sectors

Number of sectors for bank 0.

Note

The total size of bank0 sectors must be greater or equal to `bank_size`.

8.68.2.6 flash_sector_t MFSCfg::bank1_start

Base sector index for bank 1.

8.68.2.7 flash_sector_t MFSCfg::bank1_sectors

Number of sectors for bank 1.

Note

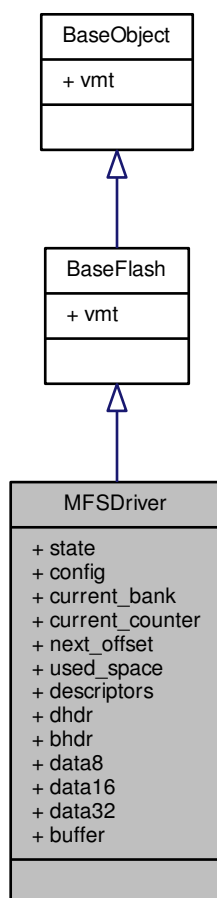
The total size of bank1 sectors must be greater or equal to `bank_size`.

8.69 MFSDriver Struct Reference

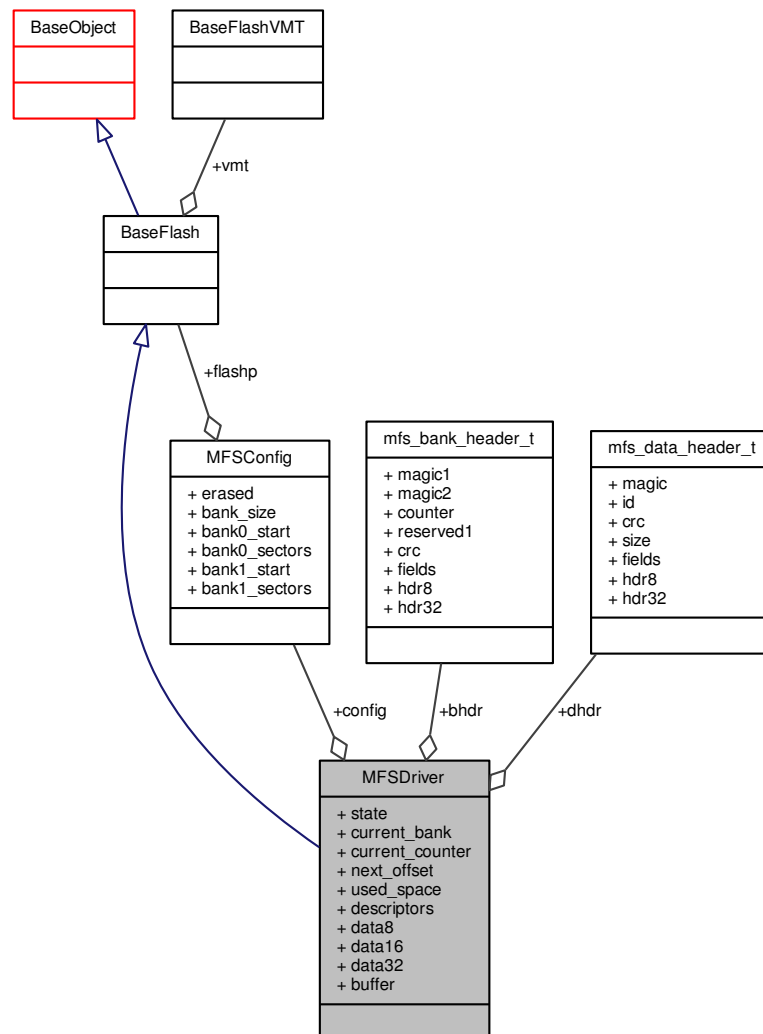
Type of an MFS instance.

```
#include <mfs.h>
```

Inheritance diagram for MFSDriver:



Collaboration diagram for MFSDriver:



Data Fields

- `mfs_state_t state`
Driver state.
- `const MFSCConfig * config`
Current configuration data.
- `mfs_bank_t current_bank`
Bank currently in use.
- `uint32_t current_counter`
Usage counter of the current bank.
- `flash_offset_t next_offset`
Pointer to the next free position in the current bank.
- `flash_offset_t used_space`
Used space in the current bank without considering erased records.

- `mfs_record_descriptor_t` [descriptors](#) [[MFS_CFG_MAX_RECORDS](#)]

Offsets of the most recent instance of the records.

- union {
 } [buffer](#)

Transient buffer.

8.69.1 Detailed Description

Type of an MFS instance.

8.69.2 Field Documentation

8.69.2.1 `mfs_state_t` `MFSDriver::state`

Driver state.

8.69.2.2 `const MFSCfg*` `MFSDriver::config`

Current configuration data.

8.69.2.3 `mfs_bank_t` `MFSDriver::current_bank`

Bank currently in use.

8.69.2.4 `uint32_t` `MFSDriver::current_counter`

Usage counter of the current bank.

8.69.2.5 `flash_offset_t` `MFSDriver::next_offset`

Pointer to the next free position in the current bank.

8.69.2.6 `flash_offset_t` `MFSDriver::used_space`

Used space in the current bank without considering erased records.

8.69.2.7 `mfs_record_descriptor_t` `MFSDriver::descriptors`[[MFS_CFG_MAX_RECORDS](#)]

Offsets of the most recent instance of the records.

Note

Zero means that there is not a record with that id.

8.69.2.8 union { ... } `MFSDriver::buffer`

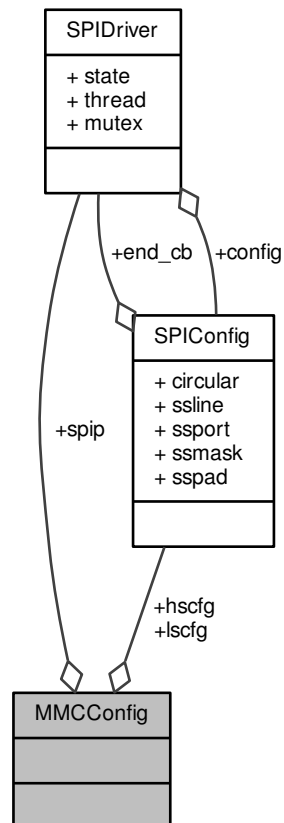
Transient buffer.

8.70 MMCCConfig Struct Reference

MMC/SD over SPI driver configuration structure.

```
#include <hal_mmc_spi.h>
```

Collaboration diagram for MMCCConfig:



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.

8.70.1 Detailed Description

MMC/SD over SPI driver configuration structure.

8.70.2 Field Documentation

8.70.2.1 `SPIDriver* MMCConfig::spip`

SPI driver associated to this MMC driver.

8.70.2.2 `const SPICongig* MMCConfig::lscfg`

SPI low speed configuration used during initialization.

8.70.2.3 `const SPICongig* MMCConfig::hscfg`

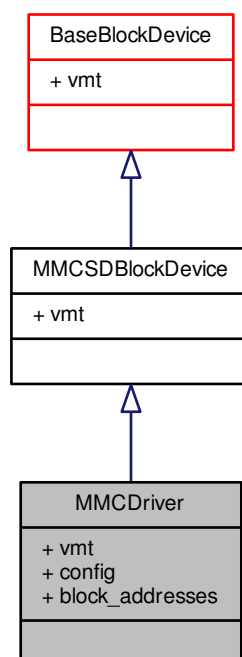
SPI high speed configuration used during transfers.

8.71 MMCDriver Struct Reference

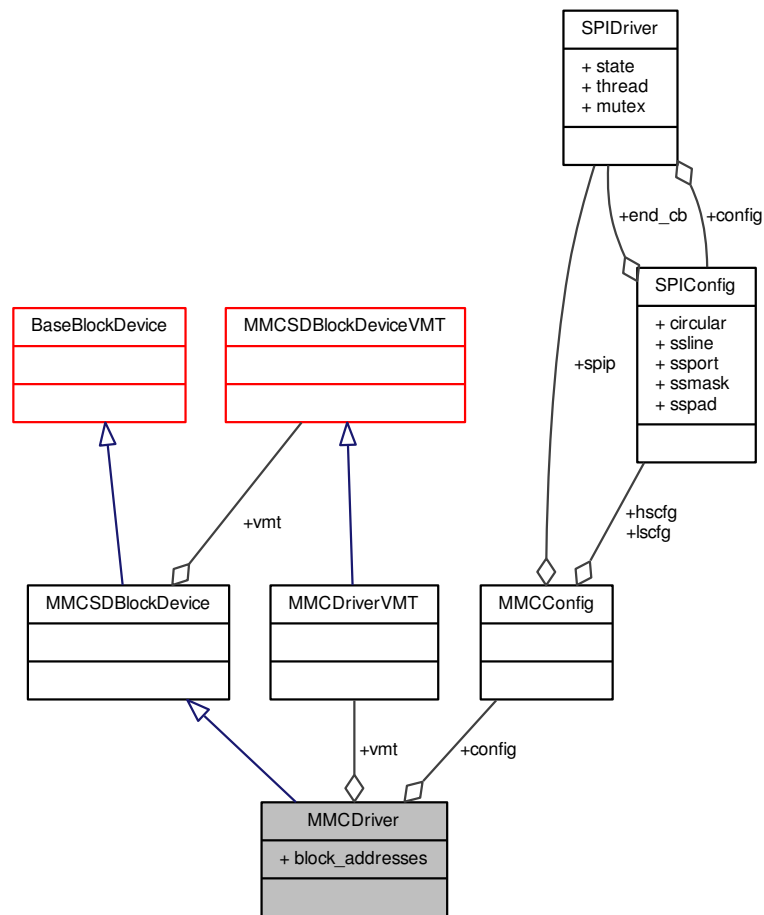
Structure representing a MMC/SD over SPI driver.

```
#include <hal_mmc_spi.h>
```

Inheritance diagram for MMCDriver:



Collaboration diagram for MMCDriver:



Data Fields

- const struct [MMCDriverVMT](#) * `vmt`
Virtual Methods Table.
- `_mmcsd_block_device_data` const [MMCCfg](#) * `config`
Current configuration data.

8.71.1 Detailed Description

Structure representing a MMC/SD over SPI driver.

8.71.2 Field Documentation

8.71.2.1 const struct [MMCDriverVMT](#)* `MMCDriver::vmt`

Virtual Methods Table.

8.71.2.2 `_mmc_sd_block_device_data` const `MMCCConfig*` `MMCDriver::config`

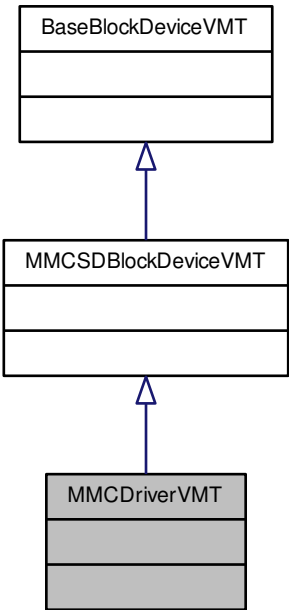
Current configuration data.

8.72 `MMCDriverVMT` Struct Reference

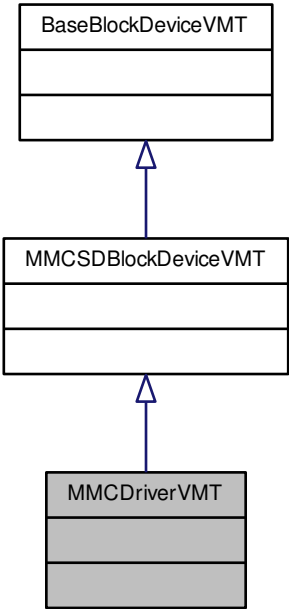
`MMCDriver` virtual methods table.

```
#include <hal_mmc_spi.h>
```

Inheritance diagram for `MMCDriverVMT`:



Collaboration diagram for MMCDriverVMT:



8.72.1 Detailed Description

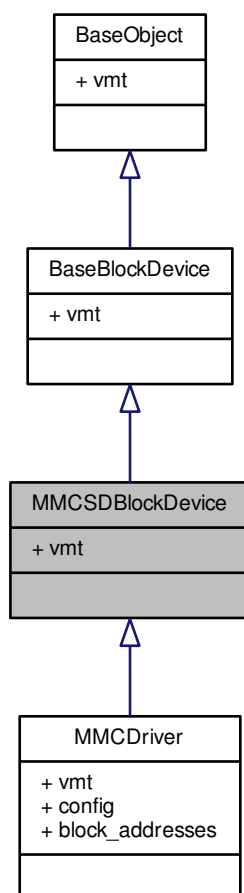
[MMCDriver](#) virtual methods table.

8.73 MMCSDBlockDevice Struct Reference

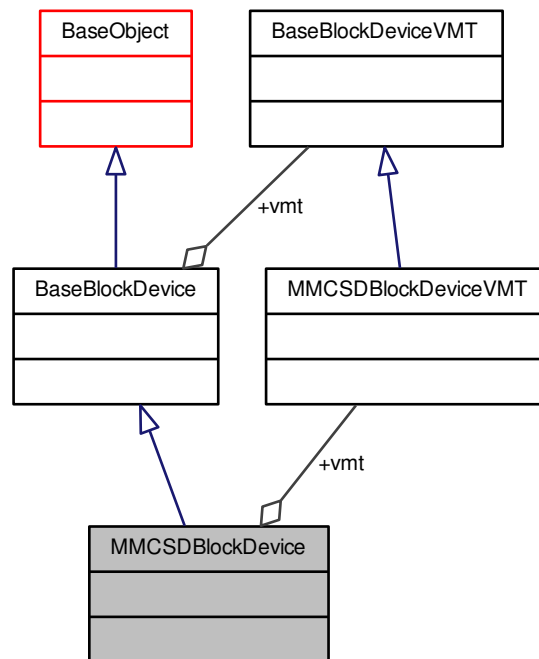
MCC/SD block device class.

```
#include <hal_mmc.h>
```

Inheritance diagram for MMCSDBlockDevice:



Collaboration diagram for MMCSDBlockDevice:



Data Fields

- const struct [MMCSDBlockDeviceVMT](#) * *vmt*

Virtual Methods Table.

8.73.1 Detailed Description

MCC/SD block device class.

This class represents a, block-accessible, MMC/SD device.

8.73.2 Field Documentation

8.73.2.1 const struct [MMCSDBlockDeviceVMT](#)* *MMCSDBlockDevice::vmt*

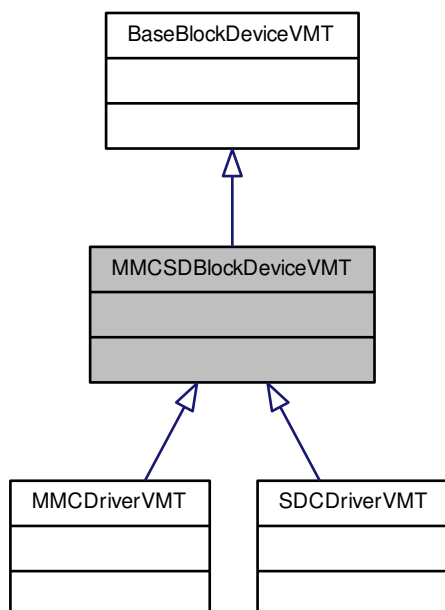
Virtual Methods Table.

8.74 MMCSDBlockDeviceVMT Struct Reference

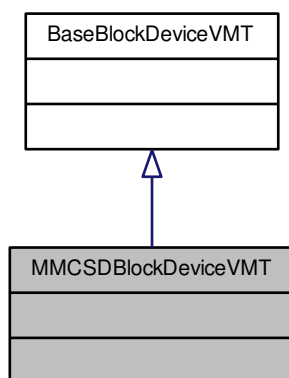
[MMCSDBlockDevice](#) virtual methods table.

```
#include <hal_mmcscd.h>
```

Inheritance diagram for MMCSDBlockDeviceVMT:



Collaboration diagram for MMCSDBlockDeviceVMT:



8.74.1 Detailed Description

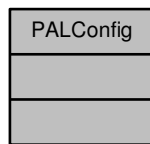
[MMCSDBlockDevice](#) virtual methods table.

8.75 PALConfig Struct Reference

Generic I/O ports static initializer.

```
#include <hal_pal_lld.h>
```

Collaboration diagram for PALConfig:



8.75.1 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

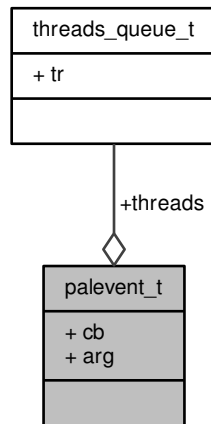
Implementations may extend this structure to contain more, architecture dependent, fields.

8.76 palevent_t Struct Reference

Type of a PAL event record.

```
#include <hal_pal.h>
```

Collaboration diagram for `palevent_t`:



Data Fields

- `threads_queue_t threads`
Threads queued for an event.
- `palcallback_t cb`
Event callback.
- `void * arg`
Event callback argument.

8.76.1 Detailed Description

Type of a PAL event record.

8.76.2 Field Documentation

8.76.2.1 `threads_queue_t palevent_t::threads`

Threads queued for an event.

8.76.2.2 `palcallback_t palevent_t::cb`

Event callback.

8.76.2.3 `void* palevent_t::arg`

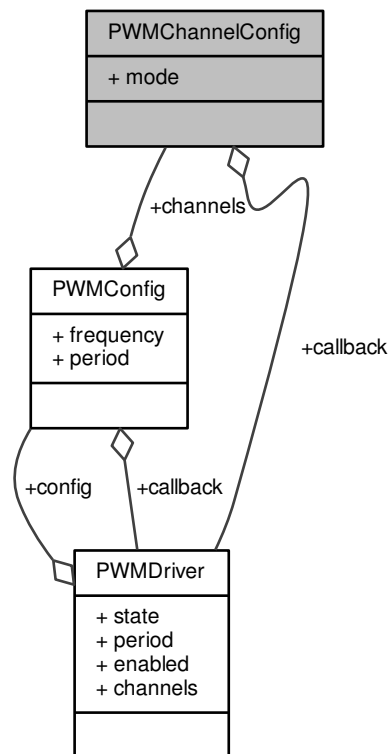
Event callback argument.

8.77 PWMChannelConfig Struct Reference

Type of a PWM driver channel configuration structure.

```
#include <hal_pwm_lld.h>
```

Collaboration diagram for PWMChannelConfig:



Data Fields

- [pwmmode_t mode](#)
Channel active logic level.
- [pwmcallback_t callback](#)
Channel callback pointer.

8.77.1 Detailed Description

Type of a PWM driver channel configuration structure.

8.77.2 Field Documentation

8.77.2.1 pwmmode_t PWMChannelConfig::mode

Channel active logic level.

8.77.2.2 pwmcallback_t PWMChannelConfig::callback

Channel callback pointer.

Note

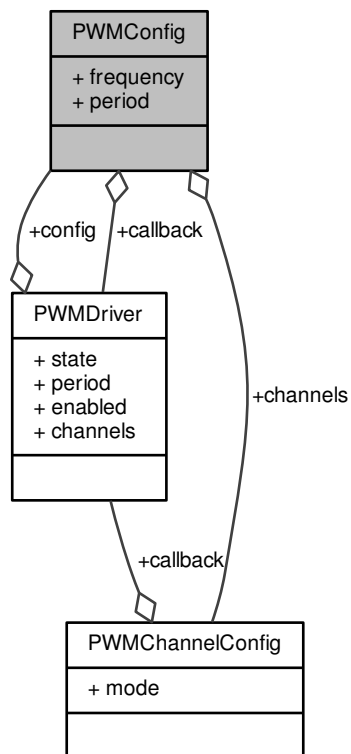
This callback is invoked on the channel compare event. If set to `NULL` then the callback is disabled.

8.78 PWMConfig Struct Reference

Type of a PWM driver configuration structure.

```
#include <hal_pwm_lld.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.

8.78.1 Detailed Description

Type of a PWM driver configuration structure.

8.78.2 Field Documentation

8.78.2.1 `uint32_t PWMConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.78.2.2 `pwmcnt_t PWMConfig::period`

PWM period in ticks.

Note

The low level can use assertions in order to catch invalid period specifications.

8.78.2.3 `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.

8.78.2.4 `PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]`

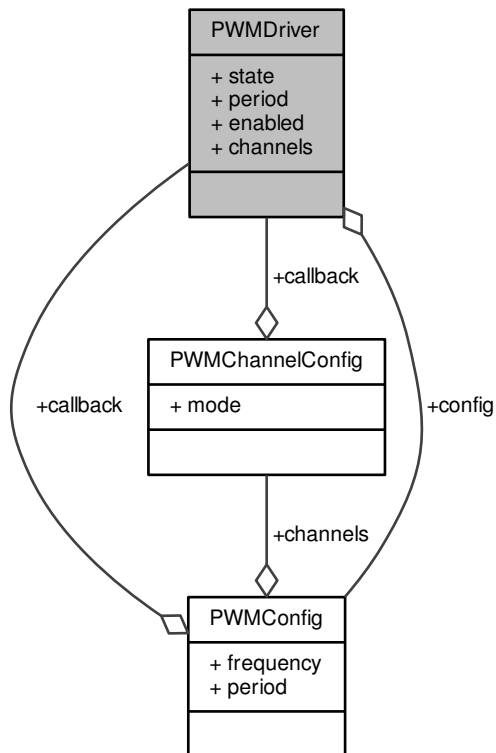
Channels configurations.

8.79 PWMDriver Struct Reference

Structure representing a PWM driver.

```
#include <hal_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.

8.79.1 Detailed Description

Structure representing a PWM driver.

8.79.2 Field Documentation

8.79.2.1 pwmstate_t PWMDriver::state

Driver state.

8.79.2.2 const PWMConfig* PWMDriver::config

Current driver configuration data.

8.79.2.3 pwmcnt_t PWMDriver::period

Current PWM period in ticks.

8.79.2.4 pwmchnmsk_t PWMDriver::enabled

Mask of the enabled channels.

8.79.2.5 pwmchannel_t PWMDriver::channels

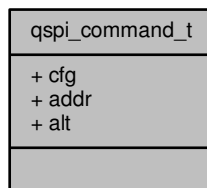
Number of channels in this instance.

8.80 qspi_command_t Struct Reference

Type of a QSPI command descriptor.

```
#include <hal_qspi.h>
```

Collaboration diagram for qspi_command_t:



8.80.1 Detailed Description

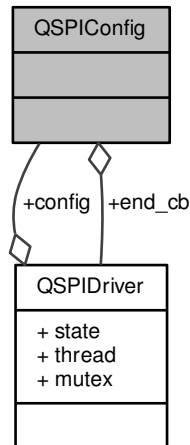
Type of a QSPI command descriptor.

8.81 QSPIConfig Struct Reference

Driver configuration structure.

```
#include <hal_qspi_llid.h>
```

Collaboration diagram for QSPIDriver:



Data Fields

- [qspicallback_t end_cb](#)

Operation complete callback or NULL.

8.81.1 Detailed Description

Driver configuration structure.

8.81.2 Field Documentation

8.81.2.1 qspicallback_t QSPIDriverConfig::end_cb

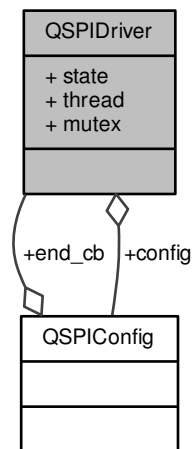
Operation complete callback or NULL.

8.82 QSPIDriver Struct Reference

Structure representing an QSPI driver.

```
#include <hal_qspi_llid.h>
```

Collaboration diagram for QSPIDriver:



Data Fields

- [qspistate_t state](#)
Driver state.
- `const QSPIDriverConfig * config`
Current configuration data.
- [thread_reference_t thread](#)
Waiting thread.
- [mutex_t mutex](#)
Mutex protecting the peripheral.

8.82.1 Detailed Description

Structure representing an QSPI driver.

8.82.2 Field Documentation

8.82.2.1 `qspistate_t QSPIDriver::state`

Driver state.

8.82.2.2 `const QSPIDriverConfig* QSPIDriver::config`

Current configuration data.

8.82.2.3 `thread_reference_t QSPIDriver::thread`

Waiting thread.

8.82.2.4 mutex_t QSPIDriver::mutex

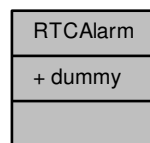
Mutex protecting the peripheral.

8.83 RTCArm Struct Reference

Type of a structure representing an RTC alarm time stamp.

```
#include <hal_rtc_lld.h>
```

Collaboration diagram for RTCArm:



8.83.1 Detailed Description

Type of a structure representing an RTC alarm time stamp.

8.84 RTCDatetime Struct Reference

Type of a structure representing an RTC date/time stamp.

```
#include <hal_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.

8.84.1 Detailed Description

Type of a structure representing an RTC date/time stamp.

8.84.2 Field Documentation

8.84.2.1 uint32_t RTCDateTime::year

Years since 1980.

8.84.2.2 uint32_t RTCDateTime::month

Months 1..12.

8.84.2.3 uint32_t RTCDateTime::dstflag

DST correction flag.

8.84.2.4 uint32_t RTCDateTime::dayofweek

Day of week 1..7.

8.84.2.5 uint32_t RTCDateTime::day

Day of the month 1..31.

8.84.2.6 uint32_t RTCDateTime::millisecond

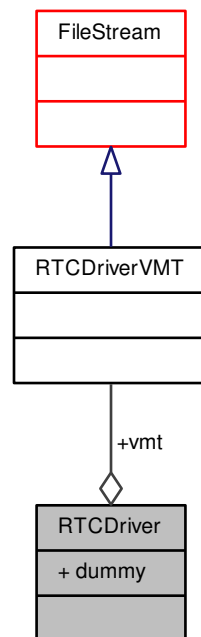
Milliseconds since midnight.

8.85 RTCDriver Struct Reference

Structure representing an RTC driver.

```
#include <hal_rtc_lld.h>
```

Collaboration diagram for RTCDriver:



Data Fields

- const struct [RTCDriverVMT](#) * [vmt](#)
Virtual Methods Table.

8.85.1 Detailed Description

Structure representing an RTC driver.

8.85.2 Field Documentation

8.85.2.1 const struct [RTCDriverVMT](#)* [RTCDriver::vmt](#)

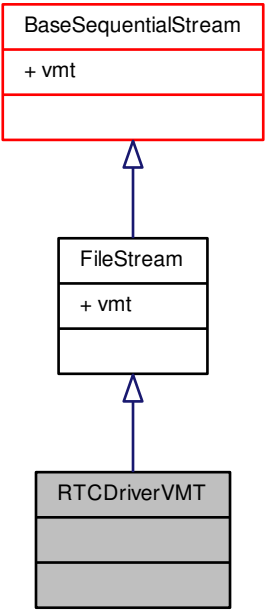
Virtual Methods Table.

8.86 RTCDriverVMT Struct Reference

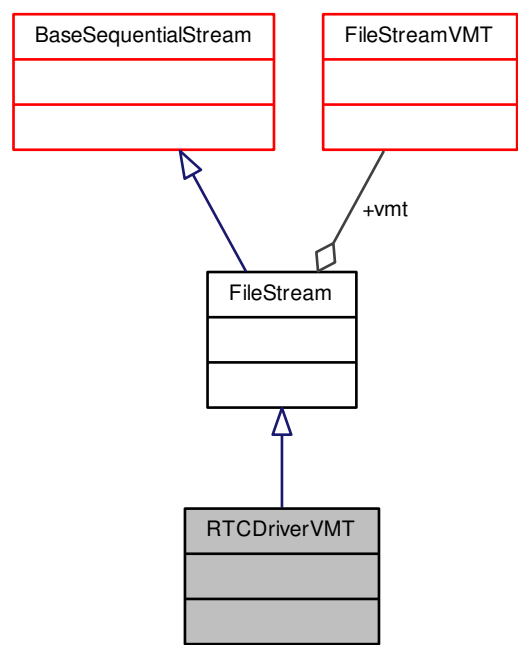
[RTCDriver](#) virtual methods table.

```
#include <hal_rtc_lld.h>
```

Inheritance diagram for RTCDriverVMT:



Collaboration diagram for RTCDriverVMT:



Additional Inherited Members

8.86.1 Detailed Description

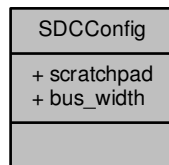
RTCDriver virtual methods table.

8.87 SDCCConfig Struct Reference

Driver configuration structure.

```
#include <hal_sdc_lld.h>
```


Collaboration diagram for SDCConfig:



Data Fields

- `uint8_t * scratchpad`
Working area for memory consuming operations.
- `sdcbusmode_t bus_width`
Bus width.

8.87.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.87.2 Field Documentation

8.87.2.1 `uint8_t* SDCConfig::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.

8.87.2.2 `sdcbusmode_t SDCConfig::bus_width`

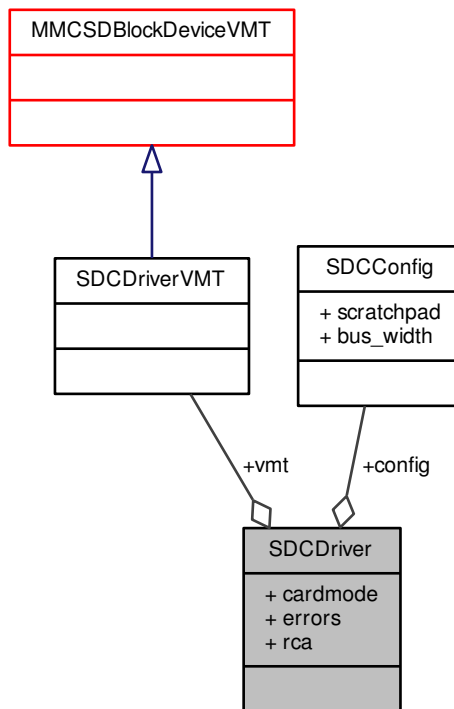
Bus width.

8.88 SDCDriver Struct Reference

Structure representing an SDC driver.

```
#include <hal_sdc_llid.h>
```

Collaboration diagram for SDCDriver:



Data Fields

- `const struct SDCDriverVMT * vmt`
Virtual Methods Table.
- `_mmc_sdb_block_device_data const SDCCConfig * config`
Current configuration data.
- `sdcmode_t cardmode`
Various flags regarding the mounted card.
- `sdclflags_t errors`
Errors flags.
- `uint32_t rca`
Card RCA.

8.88.1 Detailed Description

Structure representing an SDC driver.

8.88.2 Field Documentation

8.88.2.1 `const struct SDCDriverVMT * SDCDriver::vmt`

Virtual Methods Table.

8.88.2.2 `_mmc_sdblock_device_data` const `SDCConfig*` `SDCDriver::config`

Current configuration data.

8.88.2.3 `sdcmode_t` `SDCDriver::cardmode`

Various flags regarding the mounted card.

8.88.2.4 `sdcflags_t` `SDCDriver::errors`

Errors flags.

8.88.2.5 `uint32_t` `SDCDriver::rca`

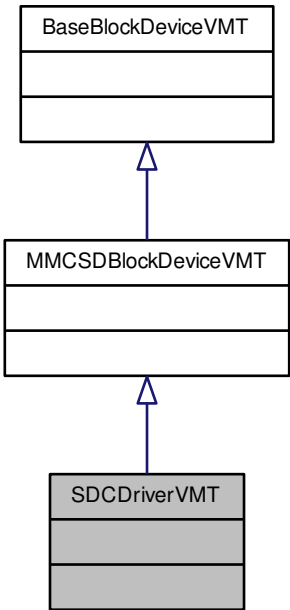
Card RCA.

8.89 SDCDriverVMT Struct Reference

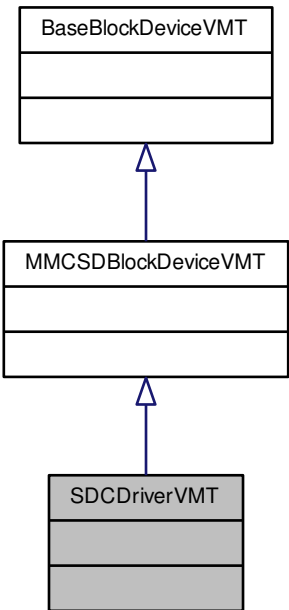
`SDCDriver` virtual methods table.

```
#include <hal_sdc_lld.h>
```

Inheritance diagram for `SDCDriverVMT`:



Collaboration diagram for SDCDriverVMT:



8.89.1 Detailed Description

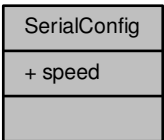
SDCDriver virtual methods table.

8.90 SerialConfig Struct Reference

PLATFORM Serial Driver configuration structure.

```
#include <hal_serial_llid.h>
```

Collaboration diagram for SerialConfig:



Data Fields

- uint32_t [speed](#)
Bit rate.

8.90.1 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.

8.90.2 Field Documentation

8.90.2.1 uint32_t SerialConfig::speed

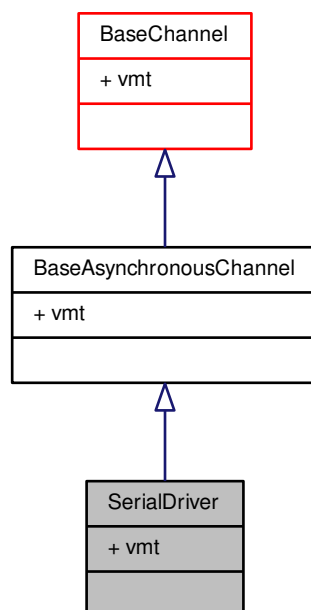
Bit rate.

8.91 SerialDriver Struct Reference

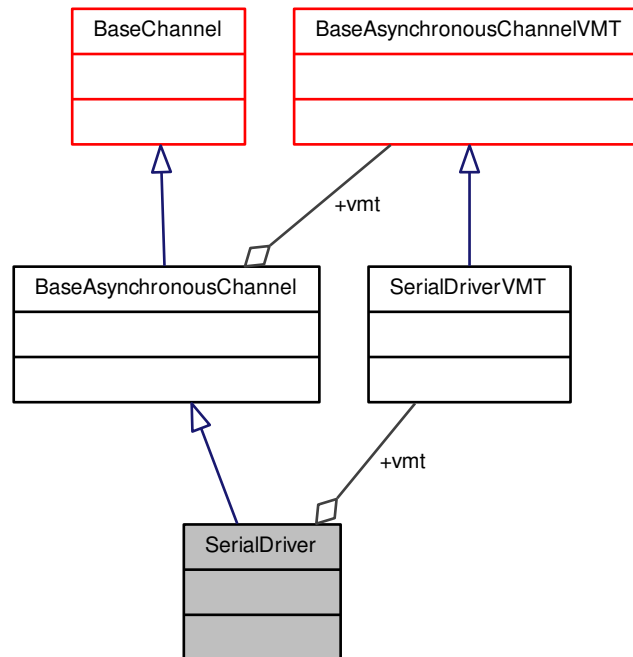
Full duplex serial driver class.

```
#include <hal_serial.h>
```

Inheritance diagram for SerialDriver:



Collaboration diagram for SerialDriver:



Data Fields

- const struct [SerialDriverVMT](#) * vmt

Virtual Methods Table.

8.91.1 Detailed Description

Full duplex serial driver class.

This class extends [BaseAsynchronousChannel](#) by adding physical I/O queues.

8.91.2 Field Documentation

8.91.2.1 const struct [SerialDriverVMT](#)* [SerialDriver::vmt](#)

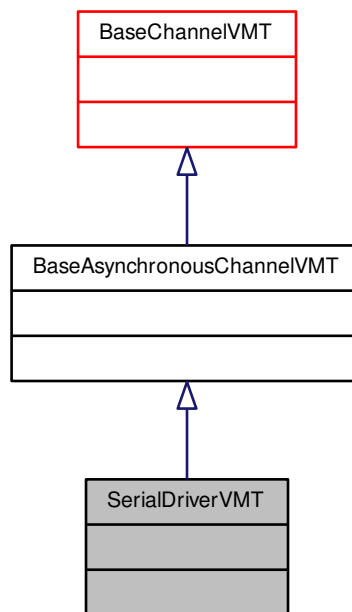
Virtual Methods Table.

8.92 SerialDriverVMT Struct Reference

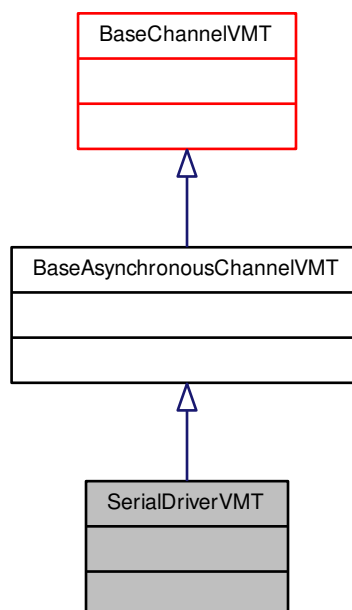
[SerialDriver](#) virtual methods table.

```
#include <hal_serial.h>
```

Inheritance diagram for SerialDriverVMT:



Collaboration diagram for SerialDriverVMT:



8.92.1 Detailed Description

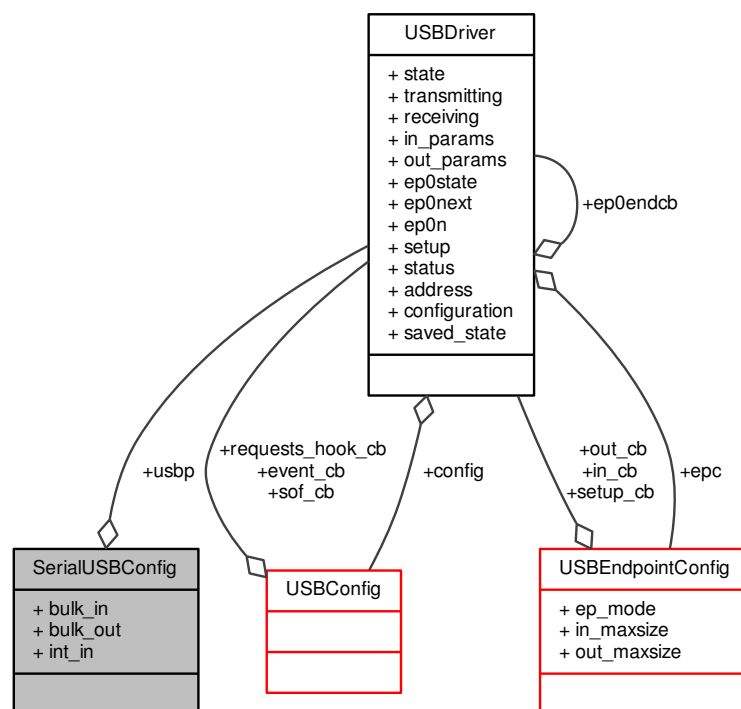
[SerialDriver](#) virtual methods table.

8.93 SerialUSBConfig Struct Reference

Serial over USB Driver configuration structure.

```
#include <hal_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.

8.93.1 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.

8.93.2 Field Documentation

8.93.2.1 USBDriver* SerialUSBConfig::usbp

USB driver to use.

8.93.2.2 usbep_t SerialUSBConfig::bulk_in

Bulk IN endpoint used for outgoing data transfer.

8.93.2.3 usbep_t SerialUSBConfig::bulk_out

Bulk OUT endpoint used for incoming data transfer.

8.93.2.4 usbep_t SerialUSBConfig::int_in

Interrupt IN endpoint used for notifications.

Note

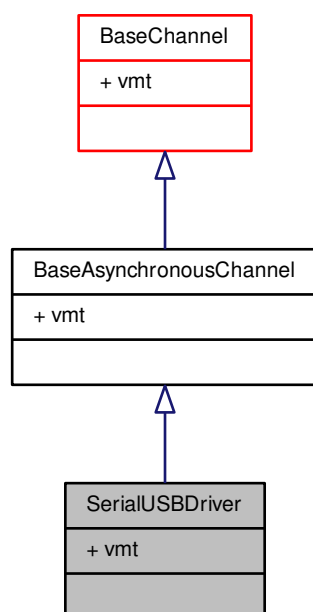
If set to zero then the INT endpoint is assumed to be not present, USB descriptors must be changed accordingly.

8.94 SerialUSBDriver Struct Reference

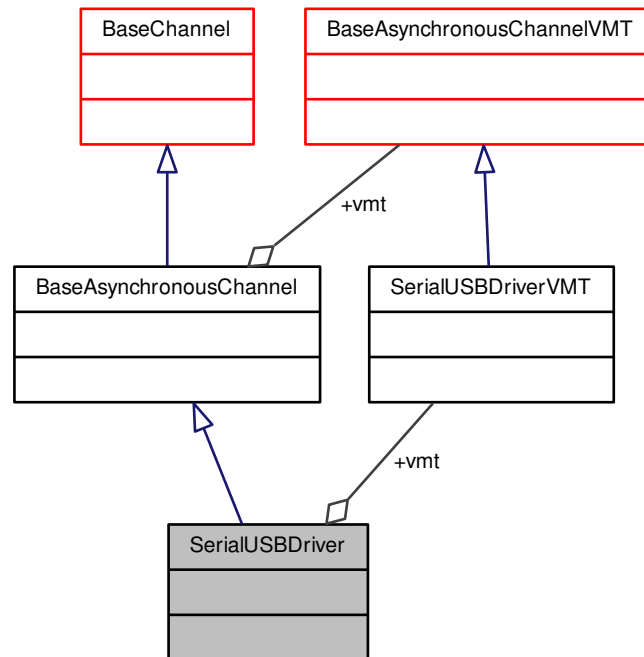
Full duplex serial driver class.

```
#include <hal_serial_usb.h>
```

Inheritance diagram for SerialUSBDriver:



Collaboration diagram for SerialUSBDriver:



Data Fields

- const struct [SerialUSBDriverVMT](#) * [vmt](#)

Virtual Methods Table.

8.94.1 Detailed Description

Full duplex serial driver class.

This class extends [BaseAsynchronousChannel](#) by adding physical I/O queues.

8.94.2 Field Documentation

8.94.2.1 const struct [SerialUSBDriverVMT](#)* [SerialUSBDriver::vmt](#)

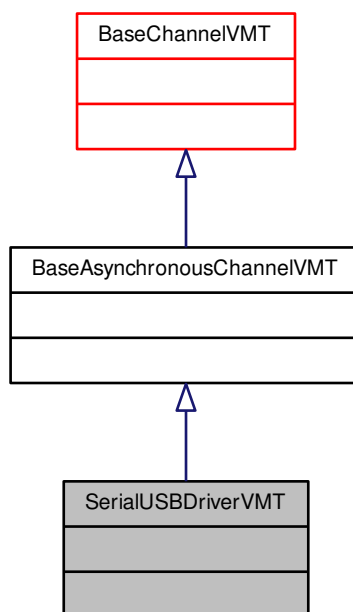
Virtual Methods Table.

8.95 SerialUSBDriverVMT Struct Reference

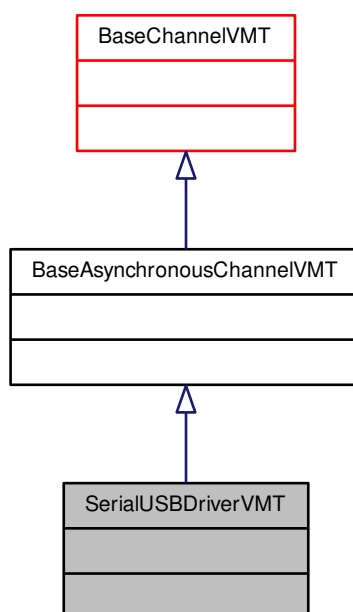
[SerialDriver](#) virtual methods table.

```
#include <hal_serial_usb.h>
```

Inheritance diagram for SerialUSBDriverVMT:



Collaboration diagram for SerialUSBDriverVMT:



8.95.1 Detailed Description

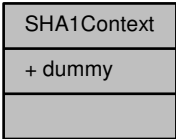
[SerialDriver](#) virtual methods table.

8.96 SHA1Context Struct Reference

Type of a SHA1 context.

```
#include <hal_crypto_lld.h>
```

Collaboration diagram for SHA1Context:



8.96.1 Detailed Description

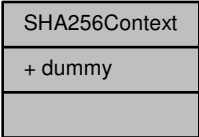
Type of a SHA1 context.

8.97 SHA256Context Struct Reference

Type of a SHA256 context.

```
#include <hal_crypto_lld.h>
```

Collaboration diagram for SHA256Context:



8.97.1 Detailed Description

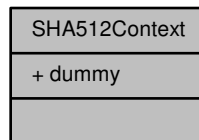
Type of a SHA256 context.

8.98 SHA512Context Struct Reference

Type of a SHA512 context.

```
#include <hal_crypto_lld.h>
```

Collaboration diagram for SHA512Context:



8.98.1 Detailed Description

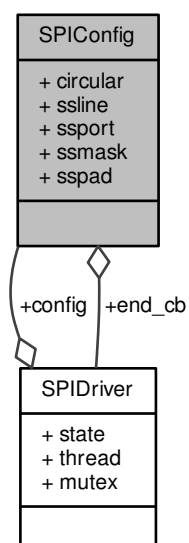
Type of a SHA512 context.

8.99 SPIConfig Struct Reference

Driver configuration structure.

```
#include <hal_spi_lld.h>
```

Collaboration diagram for SPIConfig:



Data Fields

- bool [circular](#)
Enables the circular buffer mode.
- [spicallback_t](#) [end_cb](#)
Operation complete callback or `NULL`.
- [ioline_t](#) [ssline](#)
The chip select line.
- [ioportid_t](#) [ssport](#)
The chip select port.
- [ioportmask_t](#) [ssmask](#)
The chip select port mask.
- [uint_fast8_t](#) [sspad](#)
The chip select pad number.

8.99.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.99.2 Field Documentation

8.99.2.1 bool SPIConfig::circular

Enables the circular buffer mode.

8.99.2.2 spicallback_t SPIConfig::end_cb

Operation complete callback or `NULL`.

8.99.2.3 ioline_t SPIConfig::ssline

The chip select line.

8.99.2.4 ioportid_t SPIConfig::ssport

The chip select port.

8.99.2.5 ioportmask_t SPIConfig::ssmask

The chip select port mask.

8.99.2.6 uint_fast8_t SPIConfig::sspad

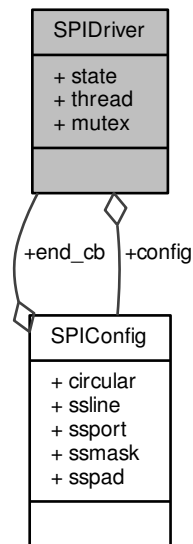
The chip select pad number.

8.100 SPIDriver Struct Reference

Structure representing an SPI driver.

```
#include <hal_spi_lld.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.

8.100.1 Detailed Description

Structure representing an SPI driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.100.2 Field Documentation

8.100.2.1 spistate_t SPIDriver::state

Driver state.

8.100.2.2 const SPIDriver::config

Current configuration data.

8.100.2.3 thread_reference_t SPIDriver::thread

Waiting thread.

8.100.2.4 mutex_t SPIDriver::mutex

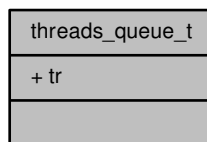
Mutex protecting the peripheral.

8.101 threads_queue_t Struct Reference

Type of a thread queue.

```
#include <osal.h>
```

Collaboration diagram for threads_queue_t:



8.101.1 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

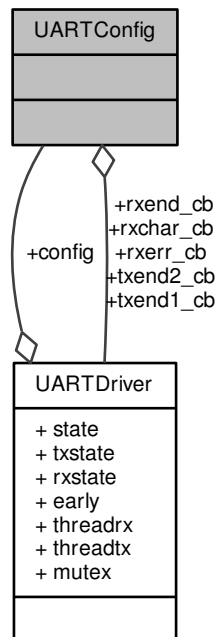
If the OSAL is implemented on a bare metal machine without RTOS then the queue can be implemented as a single thread reference.

8.102 UARTConfig Struct Reference

Driver configuration structure.

```
#include <hal_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.
- [uartecb_t rxerr_cb](#)
Receive error callback.

8.102.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.102.2 Field Documentation

8.102.2.1 `uartcb_t` `UARTConfig::txend1_cb`

End of transmission buffer callback.

8.102.2.2 `uartcb_t` `UARTConfig::txend2_cb`

Physical end of transmission callback.

8.102.2.3 `uartcb_t` `UARTConfig::rxend_cb`

Receive buffer filled callback.

8.102.2.4 `uartccb_t` `UARTConfig::rxchar_cb`

Character received while out if the `UART_RECEIVE` state.

8.102.2.5 `uartecb_t` `UARTConfig::rxerr_cb`

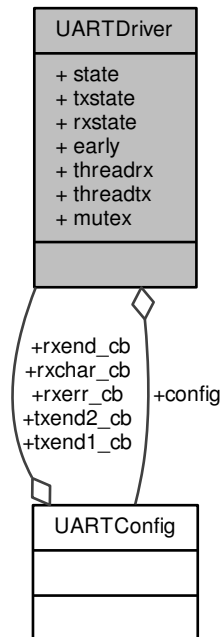
Receive error callback.

8.103 UARTDriver Struct Reference

Structure representing an UART driver.

```
#include <hal_uart_llid.h>
```

Collaboration diagram for UARTDriver:



Data Fields

- `uartstate_t state`
Driver state.
- `uarttxstate_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.

8.103.1 Detailed Description

Structure representing an UART driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.103.2 Field Documentation**8.103.2.1 uartstate_t UARTDriver::state**

Driver state.

8.103.2.2 uarttxstate_t UARTDriver::txstate

Transmitter state.

8.103.2.3 uartrxstate_t UARTDriver::rxstate

Receiver state.

8.103.2.4 const UARTConfig* UARTDriver::config

Current configuration data.

8.103.2.5 bool UARTDriver::early

Synchronization flag for transmit operations.

8.103.2.6 thread_reference_t UARTDriver::threadrx

Waiting thread on RX.

8.103.2.7 thread_reference_t UARTDriver::threaddtx

Waiting thread on TX.

8.103.2.8 mutex_t UARTDriver::mutex

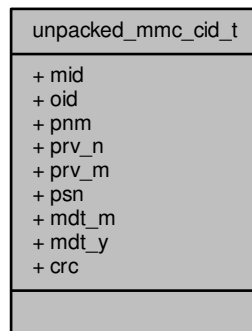
Mutex protecting the peripheral.

8.104 unpacked_mmc_cid_t Struct Reference

Unpacked CID register from MMC.

```
#include <hal_mmcsd.h>
```

Collaboration diagram for `unpacked_mmc_cid_t`:



8.104.1 Detailed Description

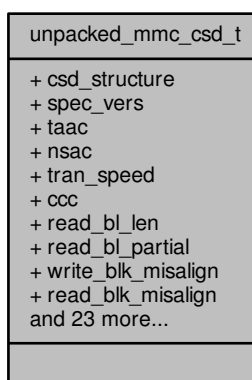
Unpacked CID register from MMC.

8.105 `unpacked_mmc_csd_t` Struct Reference

Unpacked CSD register from MMC.

```
#include <hal_mmc_csd.h>
```

Collaboration diagram for `unpacked_mmc_csd_t`:



8.105.1 Detailed Description

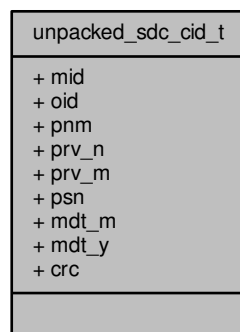
Unpacked CSD register from MMC.

8.106 unpacked_sdc_cid_t Struct Reference

Unpacked CID register from SDC.

```
#include <hal_mmcsd.h>
```

Collaboration diagram for unpacked_sdc_cid_t:



8.106.1 Detailed Description

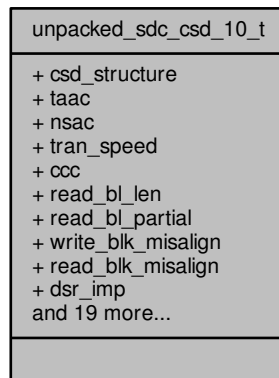
Unpacked CID register from SDC.

8.107 unpacked_sdc_csd_10_t Struct Reference

Unpacked CSD v1.0 register from SDC.

```
#include <hal_mmcsd.h>
```

Collaboration diagram for `unpacked_sdc_csd_10_t`:



8.107.1 Detailed Description

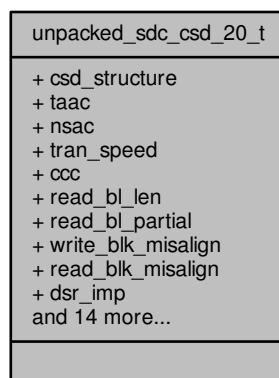
Unpacked CSD v1.0 register from SDC.

8.108 `unpacked_sdc_csd_20_t` Struct Reference

Unpacked CSD v2.0 register from SDC.

```
#include <hal_mmcscd.h>
```

Collaboration diagram for `unpacked_sdc_csd_20_t`:



8.108.1 Detailed Description

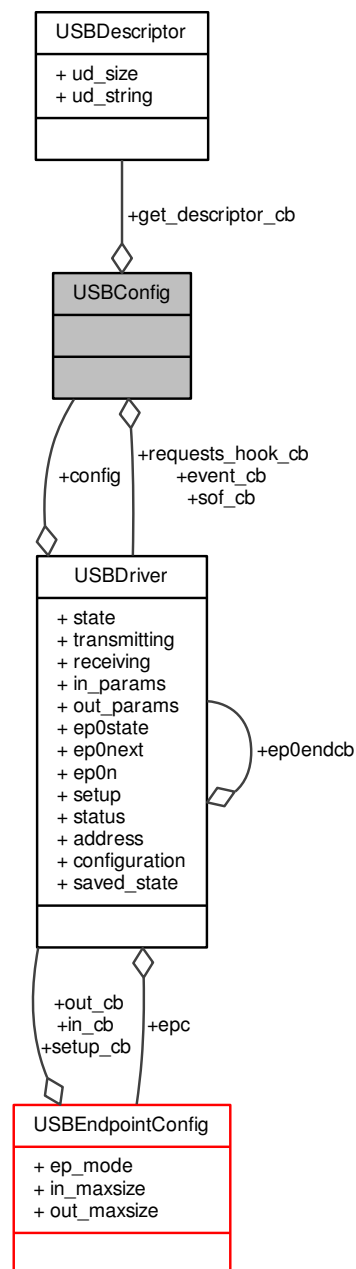
Unpacked CSD v2.0 register from SDC.

8.109 USBConfig Struct Reference

Type of an USB driver configuration structure.

```
#include <hal_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.

8.109.1 Detailed Description

Type of an USB driver configuration structure.

8.109.2 Field Documentation

8.109.2.1 usbeventcb_t USBConfig::event_cb

USB events callback.

This callback is invoked when an USB driver event is registered.

8.109.2.2 usbgetdescriptor_t USBConfig::get_descriptor_cb

Device GET_DESCRIPTOR request callback.

Note

This callback is mandatory and cannot be set to `NULL`.

8.109.2.3 usbreqhandler_t USBConfig::requests_hook_cb

Requests hook callback.

This hook allows to be notified of standard requests or to handle non standard requests.

8.109.2.4 usbcallback_t USBConfig::sof_cb

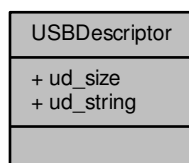
Start Of Frame callback.

8.110 USBDescriptor Struct Reference

Type of an USB descriptor.

```
#include <hal_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.

8.110.1 Detailed Description

Type of an USB descriptor.

8.110.2 Field Documentation

8.110.2.1 `size_t USBDescriptor::ud_size`

Descriptor size in unicode characters.

8.110.2.2 `const uint8_t* USBDescriptor::ud_string`

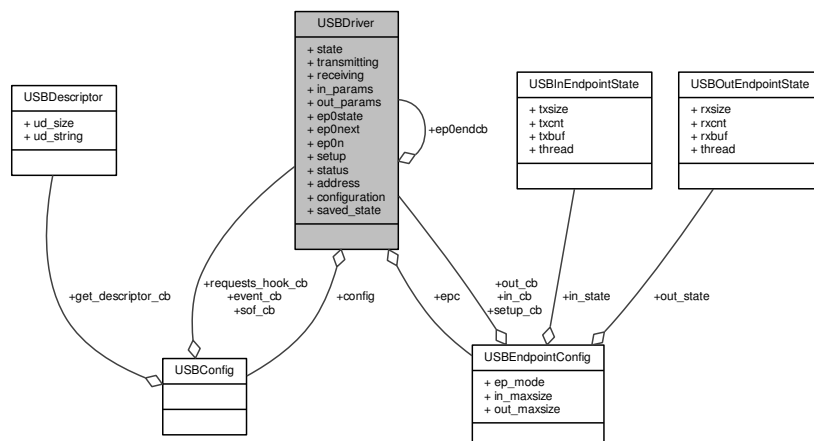
Pointer to the descriptor.

8.111 USBDriver Struct Reference

Structure representing an USB driver.

```
#include <hal_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.*
- `usbstate_t saved_state`
 - State of the driver when a suspend happened.*

8.111.1 Detailed Description

Structure representing an USB driver.

8.111.2 Field Documentation

8.111.2.1 `usbstate_t USBDriver::state`

Driver state.

8.111.2.2 `const USBConfig* USBDriver::config`

Current configuration data.

8.111.2.3 `uint16_t USBDriver::transmitting`

Bit map of the transmitting IN endpoints.

8.111.2.4 `uint16_t USBDriver::receiving`

Bit map of the receiving OUT endpoints.

8.111.2.5 `const USBEndpointConfig* USBDriver::epc[USB_MAX_ENDPOINTS+1]`

Active endpoints configurations.

8.111.2.6 `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.

8.111.2.7 `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.

8.111.2.8 `usbep0state_t USBDriver::ep0state`

Endpoint 0 state.

8.111.2.9 `uint8_t* USBDriver::ep0next`

Next position in the buffer to be transferred through endpoint 0.

8.111.2.10 `size_t USBDriver::ep0n`

Number of bytes yet to be transferred through endpoint 0.

8.111.2.11 `usbcallback_t USBDriver::ep0endcb`

Endpoint 0 end transaction callback.

8.111.2.12 `uint8_t USBDriver::setup[8]`

Setup packet buffer.

8.111.2.13 `uint16_t USBDriver::status`

Current USB device status.

8.111.2.14 `uint8_t USBDriver::address`

Assigned USB address.

8.111.2.15 `uint8_t USBDriver::configuration`

Current USB device configuration.

8.111.2.16 `usbstate_t` USBDriver::saved_state

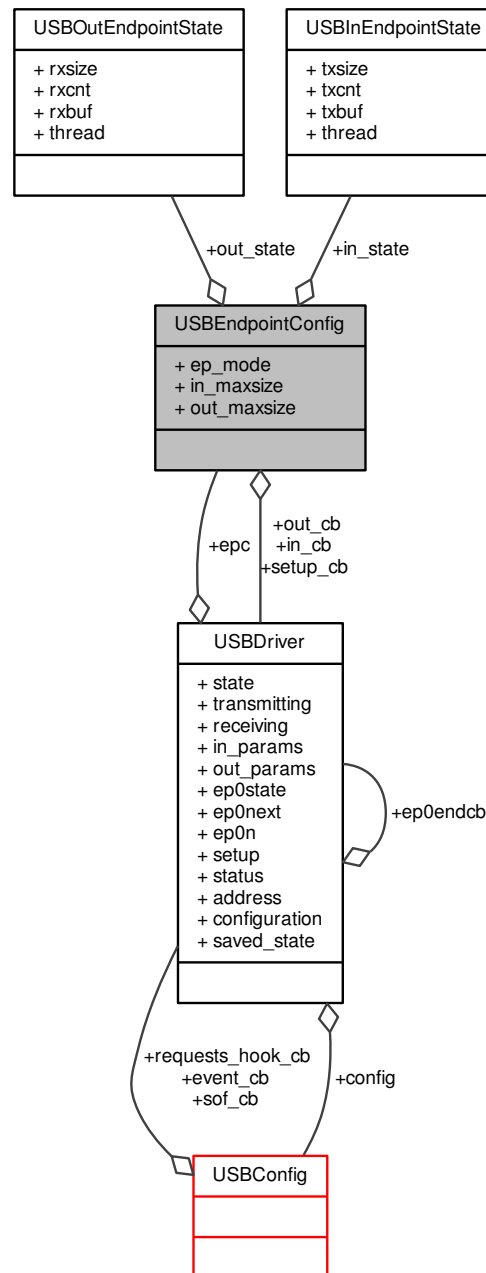
State of the driver when a suspend happened.

8.112 USBEndpointConfig Struct Reference

Type of an USB endpoint configuration structure.

```
#include <hal_usb_ll.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](#)
USBEndpointState associated to the IN endpoint.
- [USBOutEndpointState * out_state](#)
USBEndpointState associated to the OUT endpoint.

8.112.1 Detailed Description

Type of an USB endpoint configuration structure.

Note

Platform specific restrictions may apply to endpoints.

8.112.2 Field Documentation

8.112.2.1 [uint32_t USBEndpointConfig::ep_mode](#)

Type and mode of the endpoint.

8.112.2.2 [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.

8.112.2.3 [usbepcallback_t USBEndpointConfig::in_cb](#)

IN endpoint notification callback.

This field must be set to `NULL` if the IN endpoint is not used.

8.112.2.4 [usbepcallback_t USBEndpointConfig::out_cb](#)

OUT endpoint notification callback.

This field must be set to `NULL` if the OUT endpoint is not used.

8.112.2.5 uint16_t USBEndpointConfig::in_maxsize

IN endpoint maximum packet size.

This field must be set to zero if the IN endpoint is not used.

8.112.2.6 uint16_t USBEndpointConfig::out_maxsize

OUT endpoint maximum packet size.

This field must be set to zero if the OUT endpoint is not used.

8.112.2.7 USBInEndpointState* USBEndpointConfig::in_state

USBEndpointState associated to the IN endpoint.

This structure maintains the state of the IN endpoint.

8.112.2.8 USBOutEndpointState* USBEndpointConfig::out_state

USBEndpointState associated to the OUT endpoint.

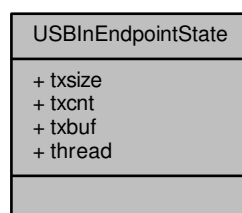
This structure maintains the state of the OUT endpoint.

8.113 USBInEndpointState Struct Reference

Type of an IN endpoint state structure.

```
#include <hal_usb_ll.h>
```

Collaboration diagram for USBInEndpointState:



Data Fields

- `size_t txsize`
Requested transmit transfer size.
- `size_t txcnt`
Transmitted bytes so far.
- `const uint8_t * txbuf`
Pointer to the transmission linear buffer.

- [thread_reference_t thread](#)

Waiting thread.

8.113.1 Detailed Description

Type of an IN endpoint state structure.

8.113.2 Field Documentation

8.113.2.1 `size_t USBInEndpointState::txsize`

Requested transmit transfer size.

8.113.2.2 `size_t USBInEndpointState::txcnt`

Transmitted bytes so far.

8.113.2.3 `const uint8_t* USBInEndpointState::txbuf`

Pointer to the transmission linear buffer.

8.113.2.4 `thread_reference_t USBInEndpointState::thread`

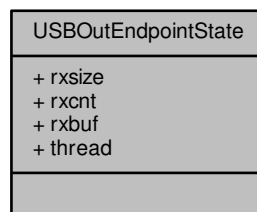
Waiting thread.

8.114 USBOutEndpointState Struct Reference

Type of an OUT endpoint state structure.

```
#include <hal_usb_ll.h>
```

Collaboration diagram for USBOutEndpointState:



Data Fields

- `size_t rxsize`

Requested receive transfer size.

- `size_t rxcnt`
Received bytes so far.
- `uint8_t * rxbuf`
Pointer to the receive linear buffer.
- `thread_reference_t thread`
Waiting thread.

8.114.1 Detailed Description

Type of an OUT endpoint state structure.

8.114.2 Field Documentation

8.114.2.1 `size_t USBOutEndpointState::rxsize`

Requested receive transfer size.

8.114.2.2 `size_t USBOutEndpointState::rxcnt`

Received bytes so far.

8.114.2.3 `uint8_t* USBOutEndpointState::rxbuf`

Pointer to the receive linear buffer.

8.114.2.4 `thread_reference_t USBOutEndpointState::thread`

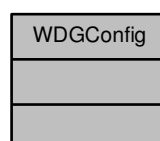
Waiting thread.

8.115 WDGConfig Struct Reference

Driver configuration structure.

```
#include <hal_wdg_lld.h>
```

Collaboration diagram for WDGConfig:



8.115.1 Detailed Description

Driver configuration structure.

Note

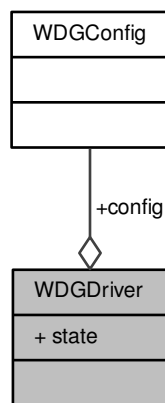
It could be empty on some architectures.

8.116 WDGDriver Struct Reference

Structure representing an WDG driver.

```
#include <hal_wdg_lld.h>
```

Collaboration diagram for WDGDriver:



Data Fields

- [wdgstate_t state](#)
Driver state.
- const [WDGConfig * config](#)
Current configuration data.

8.116.1 Detailed Description

Structure representing an WDG driver.

8.116.2 Field Documentation

8.116.2.1 wdgstate_t WDGDriver::state

Driver state.

8.116.2.2 `const WDGConfig* WDGDriver::config`

Current configuration data.

Chapter 9

File Documentation

9.1 hal.c File Reference

HAL subsystem code.

```
#include "hal.h"
```

Functions

- void [halInit](#) (void)

HAL initialization.

9.1.1 Detailed Description

HAL subsystem code.

9.2 hal.h File Reference

HAL subsystem header.

```

#include "osal.h"
#include "board.h"
#include "halconf.h"
#include "hal_lld.h"
#include "hal_objects.h"
#include "hal_streams.h"
#include "hal_channels.h"
#include "hal_files.h"
#include "hal_ioblock.h"
#include "hal_mmc_ssd.h"
#include "hal_buffers.h"
#include "hal_queues.h"
#include "hal_pal.h"
#include "hal_adc.h"
#include "hal_can.h"
#include "hal_crypto.h"
#include "hal_dac.h"
#include "hal_ext.h"
#include "hal_gpt.h"
#include "hal_i2c.h"
#include "hal_i2s.h"
#include "hal_icu.h"
#include "hal_mac.h"
#include "hal_pwm.h"
#include "hal_qspi.h"
#include "hal_rtc.h"
#include "hal_serial.h"
#include "hal_sdc.h"
#include "hal_spi.h"
#include "hal_uart.h"
#include "hal_usb.h"
#include "hal_wdg.h"
#include "hal_mmc_spi.h"
#include "hal_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 "6.0.1"`
HAL version string.
- `#define CH_HAL_MAJOR 6`
HAL version major number.
- `#define CH_HAL_MINOR 0`
HAL version minor number.
- `#define CH_HAL_PATCH 1`
HAL version patch number.

Return codes

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

Functions

- void [hallnit](#) (void)
HAL initialization.

9.2.1 Detailed Description

HAL subsystem header.

9.3 hal_accelerometer.h File Reference

Generic accelerometer interface header.

```
#include "hal_sensors.h"
```

Data Structures

- struct [BaseAccelerometerVMT](#)
BaseAccelerometer virtual methods table.
- struct [BaseAccelerometer](#)
Base accelerometer class.

Macros

- #define [_base_accelerometer_methods_alone](#)
BaseAccelerometer specific methods.
- #define [_base_accelerometer_methods](#)
BaseAccelerometer specific methods with inherited ones.
- #define [_base_accelerometer_data_base_sensor_data](#)
BaseAccelerometer specific data.

Macro Functions (BaseAccelerometer)

- #define [accelerometerGetAxesNumber](#)(ip) (ip)->vmt->get_channels_number(ip)
Accelerometer get axes number.
- #define [accelerometerReadRaw](#)(ip, dp) (ip)->vmt->read_raw(ip, dp)
Accelerometer read raw data.
- #define [accelerometerReadCooked](#)(ip, dp) (ip)->vmt->read_cooked(ip, dp)
Accelerometer read cooked data.
- #define [accelerometerSetBias](#)(ip, bp) (ip)->vmt->set_bias(ip, bp)
Updates accelerometer bias data from received buffer.
- #define [accelerometerResetBias](#)(ip) (ip)->vmt->reset_bias(ip)
Reset accelerometer bias data restoring it to zero.
- #define [accelerometerSetSensitivity](#)(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)
Updates accelerometer sensitivity data from received buffer.
- #define [accelerometerResetSensitivity](#)(ip) (ip)->vmt->reset_sensitivity(ip)
Reset accelerometer sensitivity data restoring it to its typical value.

9.3.1 Detailed Description

Generic accelerometer interface header.

9.4 hal_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 [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.

9.4.1 Detailed Description

ADC Driver code.

9.5 hal_adc.h File Reference

ADC Driver macros and structures.

```
#include "hal_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) `osalThreadResumeI`(&(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

Functions

- void `adclnit` (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 `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.

9.5.1 Detailed Description

ADC Driver macros and structures.

9.6 hal_adc_ild.c File Reference

PLATFORM ADC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [adc_ild_init](#) (void)
Low level ADC driver initialization.
- void [adc_ild_start](#) (ADCDriver *adcp)
Configures and activates the ADC peripheral.
- void [adc_ild_stop](#) (ADCDriver *adcp)
Deactivates the ADC peripheral.
- void [adc_ild_start_conversion](#) (ADCDriver *adcp)
Starts an ADC conversion.
- void [adc_ild_stop_conversion](#) (ADCDriver *adcp)
Stops an ongoing conversion.

Variables

- [ADCDriver ADCD1](#)
ADC1 driver identifier.

9.6.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.7 hal_adc_ild.h File Reference

PLATFORM ADC subsystem low level driver header.

Data Structures

- struct [ADCCConversionGroup](#)
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

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.

9.7.1 Detailed Description

PLATFORM ADC subsystem low level driver header.

9.8 hal_barometer.h File Reference

Generic barometer interface header.

```
#include "hal_sensors.h"
```

Data Structures

- struct [BaseBarometerVMT](#)
BaseBarometer virtual methods table.
- struct [BaseBarometer](#)
Base barometer class.

Macros

- `#define _base_barometer_methods_alone`
BaseBarometer specific methods.
- `#define _base_barometer_methods`
BaseBarometer specific methods with inherited ones.
- `#define _base_barometer_data _base_sensor_data`
BaseBarometer specific data.

Macro Functions (BaseBarometer)

- `#define barometerGetChannelsNumber(ip) (ip)->vmt->get_channels_number(ip)`
Barometer get channels number.
- `#define barometerReadRaw(ip, dp) (ip)->vmt->read_raw(ip, dp)`
Barometer read raw data.
- `#define barometerReadCooked(ip, dp) (ip)->vmt->read_cooked(ip, dp)`
Barometer read cooked data.
- `#define barometerSetBias(ip, bp) (ip)->vmt->set_bias(ip, bp)`
Updates barometer bias data from received buffer.
- `#define barometerResetBias(ip) (ip)->vmt->reset_bias(ip)`
Reset barometer bias data restoring it to zero.
- `#define barometerSetSensitivity(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)`
Updates barometer sensitivity data from received buffer.
- `#define barometerResetSensitivity(ip) (ip)->vmt->reset_sensitivity(ip)`
Reset barometer sensitivity data restoring it to its typical value.

9.8.1 Detailed Description

Generic barometer interface header.

9.9 hal_buffers.c File Reference

I/O Buffers code.

```
#include <string.h>
#include "hal.h"
```

Functions

- `void ibqObjectInit (input_buffers_queue_t *ibqp, bool suspended, 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, sysinterval_t timeout)`
Gets the next filled buffer from the queue.
- `msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t *ibqp, sysinterval_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, [sysinterval_t](#) timeout)
Input queue read with timeout.
- [size_t](#) [ibqReadTimeout](#) ([input_buffers_queue_t](#) *ibqp, [uint8_t](#) *bp, [size_t](#) n, [sysinterval_t](#) timeout)
Input queue read with timeout.
- void [obqObjectInit](#) ([output_buffers_queue_t](#) *obqp, bool suspended, [uint8_t](#) *bp, [size_t](#) size, [size_t](#) n, [bqnotify_t](#) only, 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, [sysinterval_t](#) timeout)
Gets the next empty buffer from the queue.
- [msg_t](#) [obqGetEmptyBufferTimeoutS](#) ([output_buffers_queue_t](#) *obqp, [sysinterval_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, [sysinterval_t](#) timeout)
Output queue write with timeout.
- [size_t](#) [obqWriteTimeout](#) ([output_buffers_queue_t](#) *obqp, const [uint8_t](#) *bp, [size_t](#) n, [sysinterval_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.

9.9.1 Detailed Description

I/O Buffers code.

9.10 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 bqSpacel(bqp) ((bqp)->bcounter)`
Return the ready buffers number.
- `#define bqGetLinkX(bqp) ((bqp)->link)`
Returns the queue application-defined link.
- `#define bqIsSuspendedX(bqp) ((bqp)->suspended)`
Return the suspended state of the queue.
- `#define bqSuspendl(bqp)`
Puts the queue in suspended state.
- `#define bqResumeX(bqp)`
Resumes normal queue operations.
- `#define ibqlsEmptyl(ibqp) ((bool)(bqSpacel(ibqp) == 0U))`
Evaluates to `true` if the specified input buffers queue is empty.
- `#define ibqlsFullI(ibqp)`
Evaluates to `true` if the specified input buffers queue is full.
- `#define obqlsEmptyl(obqp)`
Evaluates to `true` if the specified output buffers queue is empty.
- `#define obqlsFullI(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.

Functions

- `void ibqObjectInit (input_buffers_queue_t *ibqp, bool suspended, 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, sysinterval_t timeout)`
Gets the next filled buffer from the queue.
- `msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t *ibqp, sysinterval_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, [sysinterval_t](#) timeout)
Input queue read with timeout.
- [size_t](#) [ibqReadTimeout](#) ([input_buffers_queue_t](#) *ibqp, [uint8_t](#) *bp, [size_t](#) n, [sysinterval_t](#) timeout)
Input queue read with timeout.
- void [obqObjectInit](#) ([output_buffers_queue_t](#) *obqp, bool suspended, [uint8_t](#) *bp, [size_t](#) size, [size_t](#) n, [bqnotify_t](#) only, 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, [sysinterval_t](#) timeout)
Gets the next empty buffer from the queue.
- [msg_t](#) [obqGetEmptyBufferTimeoutS](#) ([output_buffers_queue_t](#) *obqp, [sysinterval_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, [sysinterval_t](#) timeout)
Output queue write with timeout.
- [size_t](#) [obqWriteTimeout](#) ([output_buffers_queue_t](#) *obqp, const [uint8_t](#) *bp, [size_t](#) n, [sysinterval_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.

9.10.1 Detailed Description

I/O Buffers macros and structures.

9.11 hal_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 `canTryReceiveI` (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
Can frame receive attempt.
- msg_t `canTransmitTimeout` (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, sysinterval_t timeout)
Can frame transmission.
- msg_t `canReceiveTimeout` (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, sysinterval_t timeout)
Can frame receive.
- void `canSleep` (CANDriver *canp)
Enters the sleep mode.
- void `canWakeup` (CANDriver *canp)
Enforces leaving the sleep mode.

9.11.1 Detailed Description

CAN Driver code.

9.12 hal_can.h File Reference

CAN Driver macros and structures.

```
#include "hal_can_lld.h"
```

Macros

- #define `CAN_ANY_MAILBOX` 0U
Special mailbox identifier.

CAN status flags

- #define `CAN_LIMIT_WARNING` 1U
Errors rate warning.
- #define `CAN_LIMIT_ERROR` 2U
Errors rate error.
- #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.
- #define `CAN_ENFORCE_USE_CALLBACKS` FALSE
Enforces the driver to use direct callbacks rather than OSAL events.

Macro Functions

- #define `CAN_MAILBOX_TO_MASK`(mbx) (1U << ((mbx) - 1U))
Converts a mailbox index to a bit mask.
- #define `canTransmit`(canp, mailbox, ctfp, timeout) `canTransmitTimeout`(canp, mailbox, ctfp, timeout)
Legacy name for `canTransmitTimeout()`.
- #define `canReceive`(canp, mailbox, crfp, timeout) `canReceiveTimeout`(canp, mailbox, crfp, timeout)
Legacy name for `canReceiveTimeout()`.

Low level driver helper macros

- #define `_can_tx_empty_isr`(canp, flags)
TX mailbox empty event.
- #define `_can_rx_full_isr`(canp, flags)
RX mailbox empty full event.
- #define `_can_wakeup_isr`(canp)
Error event.
- #define `_can_error_isr`(canp, flags)
Error event.

Enumerations

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 `canTransmitTimeout` (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, sysinterval_t timeout)
Can frame transmission.
- msg_t `canReceiveTimeout` (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, sysinterval_t timeout)
Can frame receive.

9.12.1 Detailed Description

CAN Driver macros and structures.

9.13 hal_can_ild.c File Reference

PLATFORM CAN subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [can_ild_init](#) (void)
Low level CAN driver initialization.
- void [can_ild_start](#) (CANDriver *canp)
Configures and activates the CAN peripheral.
- void [can_ild_stop](#) (CANDriver *canp)
Deactivates the CAN peripheral.
- bool [can_ild_is_tx_empty](#) (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame can be transmitted.
- void [can_ild_transmit](#) (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)
Inserts a frame into the transmit queue.
- bool [can_ild_is_rx_nonempty](#) (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame has been received.
- void [can_ild_receive](#) (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
Receives a frame from the input queue.
- void [can_ild_sleep](#) (CANDriver *canp)
Enters the sleep mode.
- void [can_ild_wakeup](#) (CANDriver *canp)
Enforces leaving the sleep mode.

Variables

- [CANDriver CAND1](#)
CAN1 driver identifier.

9.13.1 Detailed Description

PLATFORM CAN subsystem low level driver source.

9.14 hal_can_ild.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 struct `CANDriver` `CANDriver`
Type of a structure representing an CAN driver.
- typedef uint32_t `canmbx_t`
Type of a transmission mailbox index.
- typedef void(* `can_callback_t`) (`CANDriver` *canp, uint32_t flags)
Type of a CAN notification callback.

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.

9.14.1 Detailed Description

PLATFORM CAN subsystem low level driver header.

9.15 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.

Default control operation codes.

- `#define CHN_CTL_INVALID 0 /** @brief Invalid operation code. */`
- `#define CHN_CTL_NOP 1 /** @brief Does nothing. */`
- `#define CHN_CTL_TX_WAIT 2 /** @brief Wait for TX completion. */`

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.
- `#define chnControl(ip, operation, arg) ((ip)->vmt->ctl(ip, operation, arg))`
Control operation on a channel.

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.

9.15.1 Detailed Description

I/O channels access.

This header defines an abstract interface useful to access generic I/O serial devices in a standardized way.

9.16 hal_compass.h File Reference

Generic compass interface header.

```
#include "hal_sensors.h"
```

Data Structures

- struct [BaseCompassVMT](#)
BaseCompass virtual methods table.
- struct [BaseCompass](#)
Base compass class.

Macros

- #define [_base_compass_methods_alone](#)
BaseCompass specific methods.
- #define [_base_compass_methods](#)
BaseCompass specific methods with inherited ones.
- #define [_base_compass_data_base_sensor_data](#)
BaseCompass specific data.

Macro Functions (BaseCompass)

- #define [compassGetAxesNumber](#)(ip) (ip)->vmt->get_channels_number(ip)
Compass get axes number.
- #define [compassReadRaw](#)(ip, dp) (ip)->vmt->read_raw(ip, dp)
Compass read raw data.
- #define [compassReadCooked](#)(ip, dp) (ip)->vmt->read_cooked(ip, dp)
Compass read cooked data.
- #define [compassSetBias](#)(ip, bp) (ip)->vmt->set_bias(ip, bp)
Updates compass bias data from received buffer.
- #define [compassResetBias](#)(ip) (ip)->vmt->reset_bias(ip)
Reset compass bias data restoring it to zero.
- #define [compassSetSensitivity](#)(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)
Updates compass sensitivity data from received buffer.
- #define [compassResetSensitivity](#)(ip) (ip)->vmt->reset_sensitivity(ip)
Reset compass sensitivity data restoring it to its typical value.

9.16.1 Detailed Description

Generic compass interface header.

9.17 hal_crypto.c File Reference

Cryptographic Driver code.

```
#include "hal.h"
```

Functions

- void [cryInit](#) (void)
Cryptographic Driver initialization.
- void [cryObjectInit](#) (CRYDriver *cryp)
Initializes the standard part of a CRYDriver structure.
- void [cryStart](#) (CRYDriver *cryp, const CRYConfig *config)
Configures and activates the cryptographic peripheral.
- void [cryStop](#) (CRYDriver *cryp)
Deactivates the cryptographic peripheral.
- [cryerror_t](#) [cryLoadTransientKey](#) (CRYDriver *cryp, [cryalgorithm_t](#) algorithm, [size_t](#) size, const [uint8_t](#) *keyp)
Initializes the transient key for a specific algorithm.
- [cryerror_t](#) [cryEncryptAES](#) (CRYDriver *cryp, [crykey_t](#) key_id, const [uint8_t](#) *in, [uint8_t](#) *out)
Encryption of a single block using AES.
- [cryerror_t](#) [cryDecryptAES](#) (CRYDriver *cryp, [crykey_t](#) key_id, const [uint8_t](#) *in, [uint8_t](#) *out)
Decryption of a single block using AES.
- [cryerror_t](#) [cryEncryptAES_ECB](#) (CRYDriver *cryp, [crykey_t](#) key_id, [size_t](#) size, const [uint8_t](#) *in, [uint8_t](#) *out)
Encryption operation using AES-ECB.
- [cryerror_t](#) [cryDecryptAES_ECB](#) (CRYDriver *cryp, [crykey_t](#) key_id, [size_t](#) size, const [uint8_t](#) *in, [uint8_t](#) *out)
Decryption operation using AES-ECB.
- [cryerror_t](#) [cryEncryptAES_CBC](#) (CRYDriver *cryp, [crykey_t](#) key_id, [size_t](#) size, const [uint8_t](#) *in, [uint8_t](#) *out, const [uint8_t](#) *iv)
Encryption operation using AES-CBC.
- [cryerror_t](#) [cryDecryptAES_CBC](#) (CRYDriver *cryp, [crykey_t](#) key_id, [size_t](#) size, const [uint8_t](#) *in, [uint8_t](#) *out, const [uint8_t](#) *iv)
Decryption operation using AES-CBC.
- [cryerror_t](#) [cryEncryptAES_CFB](#) (CRYDriver *cryp, [crykey_t](#) key_id, [size_t](#) size, const [uint8_t](#) *in, [uint8_t](#) *out, const [uint8_t](#) *iv)
Encryption operation using AES-CFB.
- [cryerror_t](#) [cryDecryptAES_CFB](#) (CRYDriver *cryp, [crykey_t](#) key_id, [size_t](#) size, const [uint8_t](#) *in, [uint8_t](#) *out, const [uint8_t](#) *iv)
Decryption operation using AES-CFB.
- [cryerror_t](#) [cryEncryptAES_CTR](#) (CRYDriver *cryp, [crykey_t](#) key_id, [size_t](#) size, const [uint8_t](#) *in, [uint8_t](#) *out, const [uint8_t](#) *iv)
Encryption operation using AES-CTR.
- [cryerror_t](#) [cryDecryptAES_CTR](#) (CRYDriver *cryp, [crykey_t](#) key_id, [size_t](#) size, const [uint8_t](#) *in, [uint8_t](#) *out, const [uint8_t](#) *iv)
Decryption operation using AES-CTR.

- `cryerror_t cryEncryptAES_GCM` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`, `size_t aadsize`, `const uint8_t *aad`, `uint8_t *authtag`)
Encryption operation using AES-GCM.
- `cryerror_t cryDecryptAES_GCM` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`, `size_t aadsize`, `const uint8_t *aad`, `uint8_t *authtag`)
Decryption operation using AES-GCM.
- `cryerror_t cryEncryptDES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)
Encryption of a single block using (T)DES.
- `cryerror_t cryDecryptDES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)
Decryption of a single block using (T)DES.
- `cryerror_t cryEncryptDES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)
Encryption operation using (T)DES-ECB.
- `cryerror_t cryDecryptDES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)
Decryption operation using (T)DES-ECB.
- `cryerror_t cryEncryptDES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Encryption operation using (T)DES-CBC.
- `cryerror_t cryDecryptDES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)
Decryption operation using (T)DES-CBC.
- `cryerror_t crySHA1Init` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`)
Hash initialization using SHA1.
- `cryerror_t crySHA1Update` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `size_t size`, `const uint8_t *in`)
Hash update using SHA1.
- `cryerror_t crySHA1Final` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `uint8_t *out`)
Hash finalization using SHA1.
- `cryerror_t crySHA256Init` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`)
Hash initialization using SHA256.
- `cryerror_t crySHA256Update` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `size_t size`, `const uint8_t *in`)
Hash update using SHA256.
- `cryerror_t crySHA256Final` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `uint8_t *out`)
Hash finalization using SHA256.
- `cryerror_t crySHA512Init` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`)
Hash initialization using SHA512.
- `cryerror_t crySHA512Update` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `size_t size`, `const uint8_t *in`)
Hash update using SHA512.
- `cryerror_t crySHA512Final` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `uint8_t *out`)
Hash finalization using SHA512.
- `cryerror_t cryTRNG` (`CRYDriver *cryp`, `uint8_t *out`)
True random numbers generator.

9.17.1 Detailed Description

Cryptographic Driver code.

9.18 hal_crypto.h File Reference

Cryptographic Driver macros and structures.

```
#include "hal_crypto_lld.h"
#include "hal_crypto_fallback.h"
```

Data Structures

- struct [SHA1Context](#)
Type of a SHA1 context.
- struct [SHA256Context](#)
Type of a SHA256 context.
- struct [SHA512Context](#)
Type of a SHA512 context.

Macros

- #define [HAL_CRY_MAX_KEY_SIZE](#) 32
Maximum size of a key for all supported algorithms.
- #define [HAL_CRY_USE_FALLBACK](#) FALSE
Enables the SW fall-back of the cryptographic driver.
- #define [HAL_CRY_ENFORCE_FALLBACK](#) FALSE
Makes the driver forcibly use the fall-back implementations.
- #define [HAL_CRY_USE_FALLBACK](#) TRUE
Enables the SW fall-back of the cryptographic driver.

Typedefs

- typedef size_t [bitsize_t](#)
Size, in bits, of a crypto field or message.

Enumerations

Functions

- void [cryInit](#) (void)
Cryptographic Driver initialization.
- void [cryObjectInit](#) (CRYDriver *cryp)
Initializes the standard part of a CRYDriver structure.
- void [cryStart](#) (CRYDriver *cryp, const CRYConfig *config)
Configures and activates the cryptographic peripheral.
- void [cryStop](#) (CRYDriver *cryp)
Deactivates the cryptographic peripheral.
- [cryerror_t](#) [cryLoadTransientKey](#) (CRYDriver *cryp, [cryalgorithm_t](#) algorithm, size_t size, const uint8_t *keyp)
Initializes the transient key for a specific algorithm.
- [cryerror_t](#) [cryEncryptAES](#) (CRYDriver *cryp, [crykey_t](#) key_id, const uint8_t *in, uint8_t *out)
Encryption of a single block using AES.
- [cryerror_t](#) [cryDecryptAES](#) (CRYDriver *cryp, [crykey_t](#) key_id, const uint8_t *in, uint8_t *out)
Decryption of a single block using AES.

- [cryerror_t cryEncryptAES_ECB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out)
Encryption operation using AES-ECB.
- [cryerror_t cryDecryptAES_ECB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out)
Decryption operation using AES-ECB.
- [cryerror_t cryEncryptAES_CBC](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CBC.
- [cryerror_t cryDecryptAES_CBC](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Decryption operation using AES-CBC.
- [cryerror_t cryEncryptAES_CFB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CFB.
- [cryerror_t cryDecryptAES_CFB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Decryption operation using AES-CFB.
- [cryerror_t cryEncryptAES_CTR](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CTR.
- [cryerror_t cryDecryptAES_CTR](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Decryption operation using AES-CTR.
- [cryerror_t cryEncryptAES_GCM](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv, size_t aadsize, const uint8_t *aad, uint8_t *authtag)
Encryption operation using AES-GCM.
- [cryerror_t cryDecryptAES_GCM](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv, size_t aadsize, const uint8_t *aad, uint8_t *authtag)
Decryption operation using AES-GCM.
- [cryerror_t cryEncryptDES](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, const uint8_t *in, uint8_t *out)
Encryption of a single block using (T)DES.
- [cryerror_t cryDecryptDES](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, const uint8_t *in, uint8_t *out)
Decryption of a single block using (T)DES.
- [cryerror_t cryEncryptDES_ECB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out)
Encryption operation using (T)DES-ECB.
- [cryerror_t cryDecryptDES_ECB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out)
Decryption operation using (T)DES-ECB.
- [cryerror_t cryEncryptDES_CBC](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using (T)DES-CBC.
- [cryerror_t cryDecryptDES_CBC](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Decryption operation using (T)DES-CBC.
- [cryerror_t crySHA1Init](#) ([CRYDriver](#) *cryp, [SHA1Context](#) *sha1ctxp)
Hash initialization using SHA1.
- [cryerror_t crySHA1Update](#) ([CRYDriver](#) *cryp, [SHA1Context](#) *sha1ctxp, size_t size, const uint8_t *in)
Hash update using SHA1.
- [cryerror_t crySHA1Final](#) ([CRYDriver](#) *cryp, [SHA1Context](#) *sha1ctxp, uint8_t *out)
Hash finalization using SHA1.

- `cryerror_t crySHA256Init (CRYDriver *cryp, SHA256Context *sha256ctxp)`
Hash initialization using SHA256.
- `cryerror_t crySHA256Update (CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)`
Hash update using SHA256.
- `cryerror_t crySHA256Final (CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)`
Hash finalization using SHA256.
- `cryerror_t crySHA512Init (CRYDriver *cryp, SHA512Context *sha512ctxp)`
Hash initialization using SHA512.
- `cryerror_t crySHA512Update (CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)`
Hash update using SHA512.
- `cryerror_t crySHA512Final (CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)`
Hash finalization using SHA512.
- `cryerror_t cryTRNG (CRYDriver *cryp, uint8_t *out)`
True random numbers generator.

9.18.1 Detailed Description

Cryptographic Driver macros and structures.

9.19 hal_crypto_lld.c File Reference

PLATFORM cryptographic subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `cry_lld_init` (void)
Low level crypto driver initialization.
- void `cry_lld_start` (CRYDriver *cryp)
Configures and activates the crypto peripheral.
- void `cry_lld_stop` (CRYDriver *cryp)
Deactivates the crypto peripheral.
- `cryerror_t cry_lld_loadkey` (CRYDriver *cryp, cryalgorithm_t algorithm, size_t size, const uint8_t *keyp)
Initializes the transient key for a specific algorithm.
- `cryerror_t cry_lld_encrypt_AES` (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)
Encryption of a single block using AES.
- `cryerror_t cry_lld_decrypt_AES` (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)
Decryption of a single block using AES.
- `cryerror_t cry_lld_encrypt_AES_ECB` (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)
Encryption operation using AES-ECB.
- `cryerror_t cry_lld_decrypt_AES_ECB` (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)
Decryption operation using AES-ECB.
- `cryerror_t cry_lld_encrypt_AES_CBC` (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CBC.
- `cryerror_t cry_lld_decrypt_AES_CBC` (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Decryption operation using AES-CBC.

- `cryerror_t cry_ild_encrypt_AES_CFB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Encryption operation using AES-CFB.

- `cryerror_t cry_ild_decrypt_AES_CFB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Decryption operation using AES-CTR.

- `cryerror_t cry_ild_encrypt_AES_CTR` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Encryption operation using AES-CTR.

- `cryerror_t cry_ild_decrypt_AES_CTR` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Decryption operation using AES-CTR.

- `cryerror_t cry_ild_encrypt_AES_GCM` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`, `size_t aadsize`, `const uint8_t *aad`, `uint8_t *authtag`)

Encryption operation using AES-GCM.

- `cryerror_t cry_ild_decrypt_AES_GCM` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`, `size_t aadsize`, `const uint8_t *aad`, `uint8_t *authtag`)

Decryption operation using AES-GCM.

- `cryerror_t cry_ild_encrypt_DES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)

Encryption of a single block using (T)DES.

- `cryerror_t cry_ild_decrypt_DES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)

Decryption of a single block using (T)DES.

- `cryerror_t cry_ild_encrypt_DES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)

Encryption operation using (T)DES-ECB.

- `cryerror_t cry_ild_decrypt_DES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)

Decryption operation using (T)DES-ECB.

- `cryerror_t cry_ild_encrypt_DES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Encryption operation using (T)DES-CBC.

- `cryerror_t cry_ild_decrypt_DES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Decryption operation using (T)DES-CBC.

- `cryerror_t cry_ild_SHA1_init` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`)

Hash initialization using SHA1.

- `cryerror_t cry_ild_SHA1_update` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `size_t size`, `const uint8_t *in`)

Hash update using SHA1.

- `cryerror_t cry_ild_SHA1_final` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `uint8_t *out`)

Hash finalization using SHA1.

- `cryerror_t cry_ild_SHA256_init` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`)

Hash initialization using SHA256.

- `cryerror_t cry_ild_SHA256_update` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `size_t size`, `const uint8_t *in`)

Hash update using SHA256.

- `cryerror_t cry_ild_SHA256_final` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `uint8_t *out`)

Hash finalization using SHA256.

- `cryerror_t cry_ild_SHA512_init` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`)

Hash initialization using SHA512.

- `cryerror_t cry_ild_SHA512_update` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `size_t size`, `const uint8_t *in`)

Hash update using SHA512.

- `cryerror_t cry_lld_SHA512_final` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `uint8_t *out`)

Hash finalization using SHA512.

- `cryerror_t cry_lld_TRNG` (`CRYDriver *cryp`, `uint8_t *out`)

True random numbers generator.

Variables

- `CRYDriver CRYD1`

CRY1 driver identifier.

9.19.1 Detailed Description

PLATFORM cryptographic subsystem low level driver source.

9.20 hal_crypto_lld.h File Reference

PLATFORM cryptographic subsystem low level driver header.

Data Structures

- struct `CRYConfig`
Driver configuration structure.
- struct `CRYDriver`
Structure representing an CRY driver.
- struct `SHA1Context`
Type of a SHA1 context.
- struct `SHA256Context`
Type of a SHA256 context.
- struct `SHA512Context`
Type of a SHA512 context.

Macros

Driver capability switches

- `#define CRY_LLD_SUPPORTS_AES TRUE`
- `#define CRY_LLD_SUPPORTS_AES_ECB TRUE`
- `#define CRY_LLD_SUPPORTS_AES_CBC TRUE`
- `#define CRY_LLD_SUPPORTS_AES_CFB TRUE`
- `#define CRY_LLD_SUPPORTS_AES_CTR TRUE`
- `#define CRY_LLD_SUPPORTS_AES_GCM TRUE`
- `#define CRY_LLD_SUPPORTS_DES TRUE`
- `#define CRY_LLD_SUPPORTS_DES_ECB TRUE`
- `#define CRY_LLD_SUPPORTS_DES_CBC TRUE`
- `#define CRY_LLD_SUPPORTS_SHA1 TRUE`
- `#define CRY_LLD_SUPPORTS_SHA256 TRUE`
- `#define CRY_LLD_SUPPORTS_SHA512 TRUE`
- `#define CRY_LLD_SUPPORTS_TRNG TRUE`

PLATFORM configuration options

- `#define PLATFORM_CRY_USE_CRY1 FALSE`
CRY1 driver enable switch.

Typedefs

- typedef uint32_t [crykey_t](#)
CRY key identifier type.
- typedef struct [CRYDriver](#) [CRYDriver](#)
Type of a structure representing an CRY driver.

Functions

- void [cry_ild_init](#) (void)
Low level crypto driver initialization.
- void [cry_ild_start](#) ([CRYDriver](#) *cryp)
Configures and activates the crypto peripheral.
- void [cry_ild_stop](#) ([CRYDriver](#) *cryp)
Deactivates the crypto peripheral.
- [cryerror_t](#) [cry_ild_loadkey](#) ([CRYDriver](#) *cryp, [cryalgorithm_t](#) algorithm, size_t size, const uint8_t *keyp)
Initializes the transient key for a specific algorithm.
- [cryerror_t](#) [cry_ild_encrypt_AES](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, const uint8_t *in, uint8_t *out)
Encryption of a single block using AES.
- [cryerror_t](#) [cry_ild_decrypt_AES](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, const uint8_t *in, uint8_t *out)
Decryption of a single block using AES.
- [cryerror_t](#) [cry_ild_encrypt_AES_ECB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out)
Encryption operation using AES-ECB.
- [cryerror_t](#) [cry_ild_decrypt_AES_ECB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out)
Decryption operation using AES-ECB.
- [cryerror_t](#) [cry_ild_encrypt_AES_CBC](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CBC.
- [cryerror_t](#) [cry_ild_decrypt_AES_CBC](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Decryption operation using AES-CBC.
- [cryerror_t](#) [cry_ild_encrypt_AES_CFB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CFB.
- [cryerror_t](#) [cry_ild_decrypt_AES_CFB](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Decryption operation using AES-CFB.
- [cryerror_t](#) [cry_ild_encrypt_AES_CTR](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Encryption operation using AES-CTR.
- [cryerror_t](#) [cry_ild_decrypt_AES_CTR](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
Decryption operation using AES-CTR.
- [cryerror_t](#) [cry_ild_encrypt_AES_GCM](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv, size_t aadsize, const uint8_t *aad, uint8_t *authtag)
Encryption operation using AES-GCM.
- [cryerror_t](#) [cry_ild_decrypt_AES_GCM](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv, size_t aadsize, const uint8_t *aad, uint8_t *authtag)
Decryption operation using AES-GCM.
- [cryerror_t](#) [cry_ild_encrypt_DES](#) ([CRYDriver](#) *cryp, [crykey_t](#) key_id, const uint8_t *in, uint8_t *out)

Encryption of a single block using (T)DES.

- `cryerror_t cry_lld_decrypt_DES` (`CRYDriver *cryp`, `crykey_t key_id`, `const uint8_t *in`, `uint8_t *out`)

Decryption of a single block using (T)DES.

- `cryerror_t cry_lld_encrypt_DES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)

Encryption operation using (T)DES-ECB.

- `cryerror_t cry_lld_decrypt_DES_ECB` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`)

Decryption operation using (T)DES-ECB.

- `cryerror_t cry_lld_encrypt_DES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Encryption operation using (T)DES-CBC.

- `cryerror_t cry_lld_decrypt_DES_CBC` (`CRYDriver *cryp`, `crykey_t key_id`, `size_t size`, `const uint8_t *in`, `uint8_t *out`, `const uint8_t *iv`)

Decryption operation using (T)DES-CBC.

- `cryerror_t cry_lld_SHA1_init` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`)

Hash initialization using SHA1.

- `cryerror_t cry_lld_SHA1_update` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `size_t size`, `const uint8_t *in`)

Hash update using SHA1.

- `cryerror_t cry_lld_SHA1_final` (`CRYDriver *cryp`, `SHA1Context *sha1ctxp`, `uint8_t *out`)

Hash finalization using SHA1.

- `cryerror_t cry_lld_SHA256_init` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`)

Hash initialization using SHA256.

- `cryerror_t cry_lld_SHA256_update` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `size_t size`, `const uint8_t *in`)

Hash update using SHA256.

- `cryerror_t cry_lld_SHA256_final` (`CRYDriver *cryp`, `SHA256Context *sha256ctxp`, `uint8_t *out`)

Hash finalization using SHA256.

- `cryerror_t cry_lld_SHA512_init` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`)

Hash initialization using SHA512.

- `cryerror_t cry_lld_SHA512_update` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `size_t size`, `const uint8_t *in`)

Hash update using SHA512.

- `cryerror_t cry_lld_SHA512_final` (`CRYDriver *cryp`, `SHA512Context *sha512ctxp`, `uint8_t *out`)

Hash finalization using SHA512.

- `cryerror_t cry_lld_TRNG` (`CRYDriver *cryp`, `uint8_t *out`)

True random numbers generator.

9.20.1 Detailed Description

PLATFORM cryptographic subsystem low level driver header.

9.21 hal_dac.c File Reference

DAC Driver code.

```
#include "hal.h"
```


Functions

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

9.21.1 Detailed Description

DAC Driver code.

9.22 hal_dac.h File Reference

DAC Driver macros and structures.

```
#include "hal_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

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, dacsample_t *samples, size_t depth)
Starts a DAC conversion.
- void `dacStartConversionI` (DACDriver *dacp, const DACConversionGroup *grpp, 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.

9.22.1 Detailed Description

DAC Driver macros and structures.

9.23 hal_dac_1ld.c File Reference

PLATFORM DAC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [dac_ild_init](#) (void)
Low level DAC driver initialization.
- void [dac_ild_start](#) (DACDriver *dacp)
Configures and activates the DAC peripheral.
- void [dac_ild_stop](#) (DACDriver *dacp)
Deactivates the DAC peripheral.
- void [dac_ild_put_channel](#) (DACDriver *dacp, dacchannel_t channel, dacsample_t sample)
Outputs a value directly on a DAC channel.
- void [dac_ild_start_conversion](#) (DACDriver *dacp)
Starts a DAC conversion.
- void [dac_ild_stop_conversion](#) (DACDriver *dacp)
Stops an ongoing conversion.

Variables

- [DACDriver DACD1](#)
DAC1 driver identifier.

9.23.1 Detailed Description

PLATFORM DAC subsystem low level driver source.

9.24 hal_dac_ild.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, [dacsample_t](#) *buffer, size_t n)
DAC notification callback type.
- typedef void(* [dacerrorcallback_t](#)) ([DACDriver](#) *dACP, [dacerror_t](#) err)
DAC error callback type.

Enumerations

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.

9.24.1 Detailed Description

PLATFORM DAC subsystem low level driver header.

9.25 hal_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.

9.25.1 Detailed Description

EXT Driver code.

9.26 hal_ext.h File Reference

EXT Driver macros and structures.

```
#include "hal_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_LOW_LEVEL` 5U
low level 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

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.

9.26.1 Detailed Description

EXT Driver macros and structures.

9.27 hal_ext_ild.c File Reference

PLATFORM EXT subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `ext_ild_init` (void)
Low level EXT driver initialization.
- void `ext_ild_start` (EXTDriver *extp)
Configures and activates the EXT peripheral.
- void `ext_ild_stop` (EXTDriver *extp)
Deactivates the EXT peripheral.
- void `ext_ild_channel_enable` (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void `ext_ild_channel_disable` (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.

Variables

- `EXTDriver EXT1`
EXT1 driver identifier.

9.27.1 Detailed Description

PLATFORM EXT subsystem low level driver source.

9.28 hal_ext_lld.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.

9.28.1 Detailed Description

PLATFORM EXT subsystem low level driver header.

9.29 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.

9.29.1 Detailed Description

Data files.

This header defines abstract interfaces useful to access generic data files in a standardized way.

9.30 hal_flash.c File Reference

Generic flash driver class code.

```
#include "hal.h"
#include "hal_flash.h"
```

Functions

- [flash_error_t flashWaitErase](#) ([BaseFlash](#) *devp)
Waits until the current erase operation is finished.
- [flash_offset_t flashGetSectorOffset](#) ([BaseFlash](#) *devp, [flash_sector_t](#) sector)
Returns the offset of a sector.
- [uint32_t flashGetSectorSize](#) ([BaseFlash](#) *devp, [flash_sector_t](#) sector)
Returns the size of a sector.

9.30.1 Detailed Description

Generic flash driver class code.

9.31 hal_flash.h File Reference

Generic flash driver class header.

Data Structures

- struct [flash_sector_descriptor_t](#)
Flash sector descriptor.
- struct [flash_descriptor_t](#)
Type of a flash device descriptor.
- struct [BaseFlashVMT](#)
BaseFlash virtual methods table.
- struct [BaseFlash](#)
Base flash class.

Macros

- [#define _base_flash_methods_alone](#)
BaseFlash specific methods.
- [#define _base_flash_methods](#)
BaseFlash specific methods with inherited ones.
- [#define _base_flash_data](#)
BaseFlash specific data.

Flash attributes

- [#define FLASH_ATTR_ERASED_IS_ONE](#) 0x00000001
- [#define FLASH_ATTR_MEMORY_MAPPED](#) 0x00000002
- [#define FLASH_ATTR_REWRITABLE](#) 0x00000004
- [#define FLASH_ATTR_READ_ECC_CAPABLE](#) 0x00000008

- `#define FLASH_ATTR_SUSPEND_ERASE_CAPABLE 0x00000010`

Macro Functions (BaseFlash)

- `#define getBaseFlash(ip) ((BaseFlash *)&(ip)->vmt)`
Instance getter.
- `#define flashGetDescriptor(ip) (ip)->vmt->get_descriptor(ip)`
Sensors get axes number.
- `#define flashRead(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)`
Read operation.
- `#define flashProgram(ip, offset, n, pp) (ip)->vmt->program(ip, offset, n, pp)`
Program operation.
- `#define flashStartEraseAll(ip) (ip)->vmt->start_erase_all(ip)`
Starts a whole-device erase operation.
- `#define flashStartEraseSector(ip, sector) (ip)->vmt->start_erase_sector(ip, sector)`
Starts an sector erase operation.
- `#define flashQueryErase(ip, msec) (ip)->vmt->query_erase(ip, msec)`
Queries the driver for erase operation progress.
- `#define flashVerifyErase(ip, sector) (ip)->vmt->verify_erase(ip, sector)`
Returns the erase state of a sector.

Typedefs

- `typedef uint32_t flash_offset_t`
Type of a flash offset.
- `typedef uint32_t flash_sector_t`
Type of a flash sector number.

Enumerations

Functions

- `flash_error_t flashWaitErase (BaseFlash *devp)`
Waits until the current erase operation is finished.
- `flash_offset_t flashGetSectorOffset (BaseFlash *devp, flash_sector_t sector)`
Returns the offset of a sector.
- `uint32_t flashGetSectorSize (BaseFlash *devp, flash_sector_t sector)`
Returns the size of a sector.

9.31.1 Detailed Description

Generic flash driver class header.

9.32 hal_gpt.c File Reference

GPT Driver code.

```
#include "hal.h"
```

Functions

- void `gptInit` (void)
GPT Driver initialization.
- void `gptObjectInit` (GPTDriver *gtp)
Initializes the standard part of a `GPTDriver` structure.
- void `gptStart` (GPTDriver *gtp, const GPTConfig *config)
Configures and activates the GPT peripheral.
- void `gptStop` (GPTDriver *gtp)
Deactivates the GPT peripheral.
- void `gptChangeInterval` (GPTDriver *gtp, gptcnt_t interval)
Changes the interval of GPT peripheral.
- void `gptStartContinuous` (GPTDriver *gtp, gptcnt_t interval)
Starts the timer in continuous mode.
- void `gptStartContinuousl` (GPTDriver *gtp, gptcnt_t interval)
Starts the timer in continuous mode.
- void `gptStartOneShot` (GPTDriver *gtp, gptcnt_t interval)
Starts the timer in one shot mode.
- void `gptStartOneShotl` (GPTDriver *gtp, gptcnt_t interval)
Starts the timer in one shot mode.
- void `gptStopTimer` (GPTDriver *gtp)
Stops the timer.
- void `gptStopTimerl` (GPTDriver *gtp)
Stops the timer.
- void `gptPolledDelay` (GPTDriver *gtp, gptcnt_t interval)
Starts the timer in one shot mode and waits for completion.

9.32.1 Detailed Description

GPT Driver code.

9.33 hal_gpt.h File Reference

GPT Driver macros and structures.

```
#include "hal_gpt_lld.h"
```

Macros

- #define `gptChangeIntervall`(gtp, interval)
Changes the interval of GPT peripheral.
- #define `gptGetIntervalX`(gtp) gpt_lld_get_interval(gtp)
Returns the interval of GPT peripheral.
- #define `gptGetCounterX`(gtp) gpt_lld_get_counter(gtp)
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

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 [gptStartContinuousl](#) ([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.

9.33.1 Detailed Description

GPT Driver macros and structures.

9.34 hal_gpt_lld.c File Reference

PLATFORM GPT subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [gpt_ild_init](#) (void)
Low level GPT driver initialization.
- void [gpt_ild_start](#) (GPTDriver *gtp)
Configures and activates the GPT peripheral.
- void [gpt_ild_stop](#) (GPTDriver *gtp)
Deactivates the GPT peripheral.
- void [gpt_ild_start_timer](#) (GPTDriver *gtp, gptcnt_t interval)
Starts the timer in continuous mode.
- void [gpt_ild_stop_timer](#) (GPTDriver *gtp)
Stops the timer.
- void [gpt_ild_polled_delay](#) (GPTDriver *gtp, gptcnt_t interval)
Starts the timer in one shot mode and waits for completion.

Variables

- [GPTDriver GPTD1](#)
GPTD1 driver identifier.

9.34.1 Detailed Description

PLATFORM GPT subsystem low level driver source.

9.35 hal_gpt_ild.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_ild_change_interval](#)(gtp, 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.

9.35.1 Detailed Description

PLATFORM GPT subsystem low level driver header.

9.36 hal_gyroscope.h File Reference

Generic gyroscope interface header.

```
#include "hal_sensors.h"
```

Data Structures

- struct [BaseGyroscopeVMT](#)
BaseGyroscope virtual methods table.
- struct [BaseGyroscope](#)
Base gyroscope class.

Macros

- #define [_base_gyroscope_methods_alone](#)
BaseGyroscope specific methods.
- #define [_base_gyroscope_methods](#)
BaseGyroscope specific methods with inherited ones.
- #define [_base_gyroscope_data_base_sensor_data](#)
BaseGyroscope specific data.

Macro Functions (BaseGyroscope)

- #define [gyroscopeGetAxesNumber](#)(ip) (ip)->vmt->get_channels_number(ip)
Gyroscope get axes number.
- #define [gyroscopeReadRaw](#)(ip, dp) (ip)->vmt->read_raw(ip, dp)
Gyroscope read raw data.
- #define [gyroscopeReadCooked](#)(ip, dp) (ip)->vmt->read_cooked(ip, dp)
Gyroscope read cooked data.
- #define [gyroscopeSampleBias](#)(ip) (ip)->vmt->sample_bias(ip)

- Gyroscope bias sampling procedure.
 • #define [gyroscopeSetBias](#)(ip, bp) (ip)->vmt->set_bias(ip, bp)
 Updates gyroscope bias data from received buffer.
- #define [gyroscopeResetBias](#)(ip) (ip)->vmt->reset_bias(ip)
 Reset gyroscope bias data restoring it to zero.
- #define [gyroscopeSetSensitivity](#)(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)
 Updates gyroscope sensitivity data from received buffer.
- #define [gyroscopeResetSensitivity](#)(ip) (ip)->vmt->reset_sensitivity(ip)
 Reset gyroscope sensitivity data restoring it to its typical value.

9.36.1 Detailed Description

Generic gyroscope interface header.

9.37 hal_hygrometer.h File Reference

Generic hygrometer interface header.

```
#include "hal_sensors.h"
```

Data Structures

- struct [BaseHygrometerVMT](#)
[BaseHygrometer](#) virtual methods table.
- struct [BaseHygrometer](#)
 Base hygrometer class.

Macros

- #define [_base_hygrometer_methods_alone](#)
[BaseHygrometer](#) specific methods.
- #define [_base_hygrometer_methods](#)
[BaseHygrometer](#) specific methods with inherited ones.
- #define [_base_hygrometer_data_base_sensor_data](#)
[BaseHygrometer](#) specific data.

Macro Functions (BaseHygrometer)

- #define [hygrometerGetChannelsNumber](#)(ip) (ip)->vmt->get_channels_number(ip)
 Hygrometer get channels number.
- #define [hygrometerReadRaw](#)(ip, dp) (ip)->vmt->read_raw(ip, dp)
 Hygrometer read raw data.
- #define [hygrometerReadCooked](#)(ip, dp) (ip)->vmt->read_cooked(ip, dp)
 Hygrometer read cooked data.
- #define [hygrometerSetBias](#)(ip, bp) (ip)->vmt->set_bias(ip, bp)
 Updates hygrometer bias data from received buffer.
- #define [hygrometerResetBias](#)(ip) (ip)->vmt->reset_bias(ip)
 Reset hygrometer bias data restoring it to zero.
- #define [hygrometerSetSensitivity](#)(ip, sp) (ip)->vmt->set_sensitivity(ip, sp)
 Updates hygrometer sensitivity data from received buffer.
- #define [hygrometerResetSensitivity](#)(ip) (ip)->vmt->reset_sensitivity(ip)
 Reset hygrometer sensitivity data restoring it to its typical value.

9.37.1 Detailed Description

Generic hygrometer interface header.

9.38 hal_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, [sysinterval_t](#) timeout)
Sends data via the I2C bus.
- [msg_t](#) [i2cMasterReceiveTimeout](#) ([I2CDriver](#) *i2cp, [i2caddr_t](#) addr, [uint8_t](#) *rxbuf, [size_t](#) rxbytes, [sysinterval_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.

9.38.1 Detailed Description

I2C Driver code.

9.39 hal_i2c.h File Reference

I2C Driver macros and structures.

```
#include "hal_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

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, `sysinterval_t` timeout)
Sends data via the I2C bus.
- `msg_t` `i2cMasterReceiveTimeout` (I2CDriver *i2cp, `i2caddr_t` addr, `uint8_t` *rxbuf, `size_t` rxbytes, `sysinterval_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.

9.39.1 Detailed Description

I2C Driver macros and structures.

9.40 hal_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](#) *rxbuf, [size_t](#) rxbytes, [sysinterval_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](#) *rxbuf, [size_t](#) rxbytes, [sysinterval_t](#) timeout)
Transmits data via the I2C bus as master.

Variables

- [I2CDriver I2CD1](#)
I2C1 driver identifier.

9.40.1 Detailed Description

PLATFORM I2C subsystem low level driver source.

9.41 hal_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_llc_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_llc_init (void)`
Low level I2C driver initialization.
- `void i2c_llc_start (I2CDriver *i2cp)`
Configures and activates the I2C peripheral.
- `void i2c_llc_stop (I2CDriver *i2cp)`
Deactivates the I2C peripheral.
- `msg_t i2c_llc_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)`
Transmits data via the I2C bus as master.
- `msg_t i2c_llc_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)`
Receives data via the I2C bus as master.

9.41.1 Detailed Description

PLATFORM I2C subsystem low level driver header.

9.42 hal_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.

9.42.1 Detailed Description

I2S Driver code.

9.43 hal_i2s.h File Reference

I2S Driver macros and structures.

```
#include "hal_i2s_lld.h"
```

Macros

I2S modes

- #define `I2S_MODE_SLAVE` 0
- #define `I2S_MODE_MASTER` 1

Macro Functions

- #define `i2sStartExchangeI`(`i2sp`)
Starts a I2S data exchange.
- #define `i2sStopExchangeI`(`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

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.

9.43.1 Detailed Description

I2S Driver macros and structures.

9.44 hal_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.
- void [i2s_lld_stop](#) (I2SDriver *i2sp)
Deactivates the I2S peripheral.
- void [i2s_lld_start_exchange](#) (I2SDriver *i2sp)
Starts a I2S data exchange.
- void [i2s_lld_stop_exchange](#) (I2SDriver *i2sp)
Stops the ongoing data exchange.

Variables

- [I2SDriver I2SD1](#)
I2S2 driver identifier.

9.44.1 Detailed Description

PLATFORM I2S subsystem low level driver source.

9.45 hal_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_llc_init](#) (void)
Low level I2S driver initialization.
- void [i2s_llc_start](#) ([I2SDriver](#) *i2sp)
Configures and activates the I2S peripheral.
- void [i2s_llc_stop](#) ([I2SDriver](#) *i2sp)
Deactivates the I2S peripheral.
- void [i2s_llc_start_exchange](#) ([I2SDriver](#) *i2sp)
Starts a I2S data exchange.
- void [i2s_llc_stop_exchange](#) ([I2SDriver](#) *i2sp)
Stops the ongoing data exchange.

9.45.1 Detailed Description

PLATFORM I2S subsystem low level driver header.

9.46 hal_icu.c File Reference

ICU Driver code.

```
#include "hal.h"
```

Functions

- void [icuin](#) (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.

9.46.1 Detailed Description

ICU Driver code.

9.47 hal_icu.h File Reference

ICU Driver macros and structures.

```
#include "hal_icu_lld.h"
```

Macros

Macro Functions

- #define [icuStartCaptureI](#)(icup)
Starts the input capture.
- #define [icuStopCaptureI](#)(icup)
Stops the input capture.
- #define [icuEnableNotificationsI](#)(icup) [icu_lld_enable_notifications](#)(icup)
Enables notifications.
- #define [icuDisableNotificationsI](#)(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

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.

9.47.1 Detailed Description

ICU Driver macros and structures.

9.48 `hal_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.

9.48.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.49 hal_icu_ild.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_ild_get_width](#)(icup) 0
Returns the width of the latest pulse.
- #define [icu_ild_get_period](#)(icup) 0
Returns the width of the latest cycle.
- #define [icu_ild_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

Functions

- void [icu_ild_init](#) (void)
Low level ICU driver initialization.
- void [icu_ild_start](#) (ICUDriver *icup)
Configures and activates the ICU peripheral.
- void [icu_ild_stop](#) (ICUDriver *icup)
Deactivates the ICU peripheral.
- void [icu_ild_start_capture](#) (ICUDriver *icup)
Starts the input capture.
- bool [icu_ild_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.

9.49.1 Detailed Description

PLATFORM ICU subsystem low level driver header.

9.50 `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

9.50.1 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.

9.51 hal_jesd216_flash.c File Reference

JESD216 compliant flash driver class code.

```
#include "hal.h"
#include "hal_jesd216_flash.h"
```

9.51.1 Detailed Description

JESD216 compliant flash driver class code.

9.52 hal_jesd216_flash.h File Reference

JESD216 compliant flash driver class header.

```
#include "hal_flash.h"
```

Data Structures

- struct [JESD215FlashVMT](#)
JESD215Flash virtual methods table.
- struct [JESD215Flash](#)
Base flash class.

Macros

- #define [_jesd216_flash_methods_alone](#)
JESD215Flash specific methods.
- #define [_jesd216_flash_methods](#)
JESD215Flash specific methods with inherited ones.
- #define [_jesd216_flash_data_base_flash_data](#)
JESD215Flash specific data.

Common command codes

- #define **JESD216_CMD_READ_ID** 0x9FU
- #define **JESD216_CMD_READ** 0x03U
- #define **JESD216_CMD_WRITE_ENABLE** 0x06U
- #define **JESD216_CMD_WRITE_DISABLE** 0x04U
- #define **JESD216_CMD_READ_STATUS_REGISTER** 0x05U
- #define **JESD216_CMD_WRITE_STATUS_REGISTER** 0x01U
- #define **JESD216_CMD_PAGE_PROGRAM** 0x02U
- #define **JESD216_CMD_ERASE_4K** 0x20U

- #define JESD216_CMD_ERASE_BULK 0xC7U
- #define JESD216_CMD_PROGRAM_ERASE_RESUME 0x7AU
- #define JESD216_CMD_PROGRAM_ERASE_SUSPEND 0x75U
- #define JESD216_CMD_READ_OTP_ARRAY 0x4BU
- #define JESD216_CMD_PROGRAM_OTP_ARRAY 0x42U

Command options

- #define JESD216_CMD_EXTENDED_ADDRESSING 0x80000000U

Bus interface.

- #define JESD216_BUS_MODE_SPI 0U
- #define JESD216_BUS_MODE_QSPI1L 1U
- #define JESD216_BUS_MODE_QSPI2L 2U
- #define JESD216_BUS_MODE_QSPI4L 4U

Configuration options

- #define JESD216_BUS_MODE JESD216_BUS_MODE_QSPI4L
Physical transport interface.
- #define JESD216_SHARED_BUS TRUE
Shared bus switch.

9.52.1 Detailed Description

JESD216 compliant flash driver class header.

9.53 hal_ild.c File Reference

PLATFORM HAL subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [hal_ild_init](#) (void)
Low level HAL driver initialization.

9.53.1 Detailed Description

PLATFORM HAL subsystem low level driver source.

9.54 hal_ild.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.

9.54.1 Detailed Description

PLATFORM HAL subsystem low level driver header.

9.55 hal_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, sysinterval_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, sysinterval_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.

9.55.1 Detailed Description

MAC Driver code.

9.56 hal_mac.h File Reference

MAC Driver macros and structures.

```
#include "hal_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)
Enables the zero-copy API.
- #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

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, `sysinterval_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, `sysinterval_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.

9.56.1 Detailed Description

MAC Driver macros and structures.

9.57 hal_mac_lld.c File Reference

PLATFORM MAC subsystem low level driver source.

```
#include <string.h>
#include "hal.h"
#include "hal_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.

9.57.1 Detailed Description

PLATFORM MAC subsystem low level driver source.

9.58 hal_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)
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.

9.58.1 Detailed Description

PLATFORM MAC subsystem low level driver header.

9.59 hal_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_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_KSZ8081_ID` 0x00221560
- #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

9.59.1 Detailed Description

MII macros and structures.

9.60 hal_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 [recvr1](#) (MMCDriver *mmcp)
Receives a single byte response.
- static uint8_t [recvr3](#) (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 MMCCfg *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 $x^7 + x^3 + 1$).

9.60.1 Detailed Description

MMC over SPI driver code.

9.61 hal_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 _mmc_sd_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 MMCCfg *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.

9.61.1 Detailed Description

MMC over SPI driver header.

9.62 hal_mmcsd.c File Reference

MMC/SD cards common code.

```
#include "hal.h"
```

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 _mmcscd_get_capacity_ext (const uint8_t *ext_csd)`
Extract MMC card capacity from EXT_CSD.
- `void _mmcscd_unpack_sdc_cid (const MMCSDBlockDevice *sdcp, unpacked_sdc_cid_t *cidsdc)`
Unpacks SDC CID array in structure.
- `void _mmcscd_unpack_mmc_cid (const MMCSDBlockDevice *sdcp, unpacked_mmc_cid_t *cidmmc)`
Unpacks MMC CID array in structure.
- `void _mmcscd_unpack_csd_mmc (const MMCSDBlockDevice *sdcp, unpacked_mmc_csd_t *csdmmc)`
Unpacks MMC CSD array in structure.
- `void _mmcscd_unpack_csd_v10 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_10_t *csd10)`
Unpacks SDC CSD v1.0 array in structure.
- `void _mmcscd_unpack_csd_v20 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_20_t *csd20)`
Unpacks SDC CSD v2.0 array in structure.

9.62.1 Detailed Description

MMC/SD cards common code.

9.63 hal_mmcscd.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 _mmcscd_block_device_methods_base_block_device_methods`
MMCSDBlockDevice specific methods.
- `#define _mmcscd_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_BLK_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_MMC_READ_BLK_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_MMC_CRC_SLICE** 7U,1U
- #define **MMCSD_CID_MMC_MDT_Y_SLICE** 11U,8U
- #define **MMCSD_CID_MMC_MDT_M_SLICE** 15U,12U
- #define **MMCSD_CID_MMC_PSN_SLICE** 47U,16U
- #define **MMCSD_CID_MMC_PRV_M_SLICE** 51U,48U
- #define **MMCSD_CID_MMC_PRV_N_SLICE** 55U,52U
- #define **MMCSD_CID_MMC_PNM0_SLICE** 63U,56U
- #define **MMCSD_CID_MMC_PNM1_SLICE** 71U,64U
- #define **MMCSD_CID_MMC_PNM2_SLICE** 79U,72U
- #define **MMCSD_CID_MMC_PNM3_SLICE** 87U,80U
- #define **MMCSD_CID_MMC_PNM4_SLICE** 95U,88U
- #define **MMCSD_CID_MMC_PNM5_SLICE** 103U,96U
- #define **MMCSD_CID_MMC_OID_SLICE** 119U,104U
- #define **MMCSD_CID_MMC_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 **mmcstdGetCardCapacity**(ip) ((ip)->capacity)
Returns the card capacity in blocks.

Functions

- `uint32_t _mmcscd_get_slice` (const `uint32_t` *data, `uint32_t` end, `uint32_t` start)
Gets a bit field from a words array.
- `uint32_t _mmcscd_get_capacity` (const `uint32_t` *csd)
Extract card capacity from a CSD.
- `uint32_t _mmcscd_get_capacity_ext` (const `uint8_t` *ext_csd)
Extract MMC card capacity from EXT_CSD.
- `void _mmcscd_unpack_sdc_cid` (const `MMCSDBlockDevice` *sdcp, `unpacked_sdc_cid_t` *cidsdc)
Unpacks SDC CID array in structure.
- `void _mmcscd_unpack_mmc_cid` (const `MMCSDBlockDevice` *sdcp, `unpacked_mmc_cid_t` *cidmmc)
Unpacks MMC CID array in structure.
- `void _mmcscd_unpack_csd_mmc` (const `MMCSDBlockDevice` *sdcp, `unpacked_mmc_csd_t` *csdmmc)
Unpacks MMC CSD array in structure.
- `void _mmcscd_unpack_csd_v10` (const `MMCSDBlockDevice` *sdcp, `unpacked_sdc_csd_10_t` *csd10)
Unpacks SDC CSD v1.0 array in structure.
- `void _mmcscd_unpack_csd_v20` (const `MMCSDBlockDevice` *sdcp, `unpacked_sdc_csd_20_t` *csd20)
Unpacks SDC CSD v2.0 array in structure.

9.63.1 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.

9.64 hal_objects.h File Reference

Base object.

Data Structures

- struct `BaseObjectVMT`
BaseObject virtual methods table.
- struct `BaseObject`
Base stream class.

Macros

- `#define _base_object_methods`
BaseObject specific methods.
- `#define _base_object_data`
BaseObject specific data.

Macro Functions (BaseObject)

- `#define objGetInstance`(type, ip) (type)(((size_t)(ip)) - (ip)->vmt->instance_offset)
Returns the instance pointer starting from an interface pointer.

9.64.1 Detailed Description

Base object.

This header defines a base object that is the root for the inheritance system.

9.65 hal_pal.c File Reference

I/O Ports Abstraction Layer code.

```
#include "hal.h"
```

Functions

- [ioportmask_t palReadBus](#) (const [IOBus](#) *bus)
Read from an I/O bus.
- void [palWriteBus](#) (const [IOBus](#) *bus, [ioportmask_t](#) bits)
Write to an I/O bus.
- void [palSetBusMode](#) (const [IOBus](#) *bus, [iomode_t](#) mode)
Programs a bus with the specified mode.
- void [palSetPadCallbackI](#) ([ioportid_t](#) port, [iopadid_t](#) pad, [palcallback_t](#) cb, void *arg)
Associates a callback to a port/pad.
- void [palSetLineCallbackI](#) ([ioline_t](#) line, [palcallback_t](#) cb, void *arg)
Associates a callback to a line.
- [msg_t palWaitPadTimeoutS](#) ([ioportid_t](#) port, [iopadid_t](#) pad, [sysinterval_t](#) timeout)
Waits for an edge on the specified port/pad.
- [msg_t palWaitPadTimeout](#) ([ioportid_t](#) port, [iopadid_t](#) pad, [sysinterval_t](#) timeout)
Waits for an edge on the specified port/pad.
- [msg_t palWaitLineTimeoutS](#) ([ioline_t](#) line, [sysinterval_t](#) timeout)
Waits for an edge on the specified line.
- [msg_t palWaitLineTimeout](#) ([ioline_t](#) line, [sysinterval_t](#) timeout)
Waits for an edge on the specified line.

9.65.1 Detailed Description

I/O Ports Abstraction Layer code.

9.66 hal_pal.h File Reference

I/O Ports Abstraction Layer macros, types and structures.

```
#include "hal_pal_lld.h"
```

Data Structures

- struct [palevent_t](#)
Type of a PAL event record.
- 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_PUSHPULL 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.

PAL event modes

- `#define PAL_EVENT_MODE_EDGES_MASK 3U`
Mask of edges field.
- `#define PAL_EVENT_MODE_DISABLED 0U`
Channel disabled.
- `#define PAL_EVENT_MODE_RISING_EDGE 1U`
Rising edge callback.
- `#define PAL_EVENT_MODE_FALLING_EDGE 2U`
Falling edge callback.
- `#define PAL_EVENT_MODE_BOTH_EDGES 3U`
Both edges callback.

PAL configuration options

- `#define PAL_USE_CALLBACKS TRUE`
Enables synchronous APIs.
- `#define PAL_USE_WAIT TRUE`
Enables synchronous APIs.

Low level driver helper macros

- #define `_pal_init_event(e)`
Initializes a PAL event object.
- #define `_pal_init_event(e)`
Initializes a PAL event object.
- #define `_pal_init_event(e)`
Initializes a PAL event object.
- #define `_pal_clear_event(e)`
Clears a PAL event object.
- #define `_pal_clear_event(e)`
Clears a PAL event object.
- #define `_pal_clear_event(e)`
Clears a PAL event object.
- #define `_pal_isr_code(e)`
Common ISR code.
- #define `_pal_isr_code(e)`
Common ISR code.
- #define `_pal_isr_code(e)`
Common ISR code.

Macro Functions

- #define `palInit()` `pal_ild_init()`
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.
- `#define palEnablePadEventI(port, pad, mode)`
Pad event enable.
- `#define palDisablePadEventI(port, pad)`
Pad event disable.
- `#define palEnablePadEvent(port, pad, mode)`
Pad event enable.
- `#define palDisablePadEvent(port, pad)`
Pad event disable.
- `#define palEnableLineEventI(line, mode) palEnablePadEventI(PAL_PORT(line), PAL_PAD(line), mode)`
Line event enable.
- `#define palDisableLineEventI(line) palDisablePadEventI(PAL_PORT(line), PAL_PAD(line))`
Line event disable.
- `#define palEnableLineEvent(line, mode)`
Line event enable.
- `#define palDisableLineEvent(line)`
Line event disable.
- `#define palSetPadCallback(port, pad, cb, arg)`
Associates a callback to a pad.
- `#define palSetLineCallback(line, cb, arg)`
Associates a callback to a line.

Typedefs

- `typedef void(* palcallback_t) (void *arg)`
Type of a PAL event callback.

Functions

- `ioportmask_t palReadBus (const IOBus *bus)`
Read from an I/O bus.
- `void palWriteBus (const IOBus *bus, ioportmask_t bits)`
Write to an I/O bus.
- `void palSetBusMode (const IOBus *bus, iomode_t mode)`
Programs a bus with the specified mode.
- `void palSetPadCallbackI (ioportid_t port, iopadid_t pad, palcallback_t cb, void *arg)`
Associates a callback to a port/pad.
- `void palSetLineCallbackI (ioline_t line, palcallback_t cb, void *arg)`
Associates a callback to a line.
- `msg_t palWaitPadTimeoutS (ioportid_t port, iopadid_t pad, sysinterval_t timeout)`
Waits for an edge on the specified port/pad.
- `msg_t palWaitPadTimeout (ioportid_t port, iopadid_t pad, sysinterval_t timeout)`
Waits for an edge on the specified port/pad.
- `msg_t palWaitLineTimeoutS (ioline_t line, sysinterval_t timeout)`
Waits for an edge on the specified line.
- `msg_t palWaitLineTimeout (ioline_t line, sysinterval_t timeout)`
Waits for an edge on the specified line.

9.66.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures.

9.67 hal_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.

9.67.1 Detailed Description

PLATFORM PAL subsystem low level driver source.

9.68 hal_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.
- #define `pal_lld_get_pad_event`(port, pad) &`_pal_events`[0]; (void)(port); (void)pad
Returns a PAL event structure associated to a pad.
- #define `pal_lld_get_line_event`(line) &`_pal_events`[0]; (void)line
Returns a PAL event structure associated to a line.

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.
- typedef `uint32_t` `iopadid_t`
Type of an pad 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.

9.68.1 Detailed Description

PLATFORM PAL subsystem low level driver header.

9.69 hal_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.

9.69.1 Detailed Description

PWM Driver code.

9.70 hal_pwm.h File Reference

PWM Driver macros and structures.

```
#include "hal_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 `pwmChangePeriodl`(pwmp, value)
Changes the period the PWM peripheral.
- #define `pwmEnableChannell`(pwmp, channel, width)
Enables a PWM channel.
- #define `pwmDisableChannell`(pwmp, channel)
Disables a PWM channel.
- #define `pwmIsChannelEnabledl`(pwmp, channel) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)
Returns a PWM channel status.
- #define `pwmEnablePeriodicNotificationl`(pwmp) `pwm_lld_enable_periodic_notification`(pwmp)
Enables the periodic activation edge notification.
- #define `pwmDisablePeriodicNotificationl`(pwmp) `pwm_lld_disable_periodic_notification`(pwmp)
Disables the periodic activation edge notification.
- #define `pwmEnableChannelNotificationl`(pwmp, channel) `pwm_lld_enable_channel_notification`(pwmp, channel)
Enables a channel de-activation edge notification.
- #define `pwmDisableChannelNotificationl`(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

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.

9.70.1 Detailed Description

PWM Driver macros and structures.

9.71 hal_pwm_ild.c File Reference

PLATFORM PWM subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `pwm_ild_init` (void)
Low level PWM driver initialization.
- void `pwm_ild_start` (PWMDriver *pwmp)
Configures and activates the PWM peripheral.
- void `pwm_ild_stop` (PWMDriver *pwmp)
Deactivates the PWM peripheral.
- void `pwm_ild_enable_channel` (PWMDriver *pwmp, pwmchannel_t channel, pwmcnt_t width)
Enables a PWM channel.
- void `pwm_ild_disable_channel` (PWMDriver *pwmp, pwmchannel_t channel)
Disables a PWM channel and its notification.

- void [pwm_ild_enable_periodic_notification](#) ([PWMDriver](#) *pwmp)
Enables the periodic activation edge notification.
- void [pwm_ild_disable_periodic_notification](#) ([PWMDriver](#) *pwmp)
Disables the periodic activation edge notification.
- void [pwm_ild_enable_channel_notification](#) ([PWMDriver](#) *pwmp, [pwmchannel_t](#) channel)
Enables a channel de-activation edge notification.
- void [pwm_ild_disable_channel_notification](#) ([PWMDriver](#) *pwmp, [pwmchannel_t](#) channel)
Disables a channel de-activation edge notification.

Variables

- [PWMDriver PWMD1](#)
PWMD1 driver identifier.

9.71.1 Detailed Description

PLATFORM PWM subsystem low level driver source.

9.72 hal_pwm_ild.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_ild_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 [pwmmode_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)
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.

9.72.1 Detailed Description

PLATFORM PWM subsystem low level driver header.

9.73 hal_qspi.c File Reference

QSPI Driver code.

```
#include "hal.h"
```

Functions

- void [qspiInit](#) (void)
QSPI Driver initialization.
- void [qspiObjectInit](#) (QSPIDriver *qspip)
Initializes the standard part of a [QSPIDriver](#) structure.
- void [qspiStart](#) (QSPIDriver *qspip, const [QSPICfg](#) *config)

- Configures and activates the QSPI peripheral.*

 - void `qspiStop` (`QSPIDriver` *qspip)

Deactivates the QSPI peripheral.
- void `qspiStartCommand` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp)

Sends a command without data phase.
- void `qspiStartSend` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, size_t n, const uint8_t *txbuf)

Sends a command with data over the QSPI bus.
- void `qspiStartReceive` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, size_t n, uint8_t *rxbuf)

Sends a command then receives data over the QSPI bus.
- void `qspiCommand` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp)

Sends a command without data phase.
- void `qspiSend` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, size_t n, const uint8_t *txbuf)

Sends a command with data over the QSPI bus.
- void `qspiReceive` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, size_t n, uint8_t *rxbuf)

Sends a command then receives data over the QSPI bus.
- void `qspiMapFlash` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, uint8_t **addrp)

Maps in memory space a QSPI flash device.
- void `qspiUnmapFlash` (`QSPIDriver` *qspip)

Unmaps from memory space a QSPI flash device.
- void `qspiAcquireBus` (`QSPIDriver` *qspip)

Gains exclusive access to the QSPI bus.
- void `qspiReleaseBus` (`QSPIDriver` *qspip)

Releases exclusive access to the QSPI bus.

9.73.1 Detailed Description

QSPI Driver code.

9.74 hal_qspi.h File Reference

QSPI Driver macros and structures.

```
#include "hal_qspi_lld.h"
```

Data Structures

- struct `qspi_command_t`
- Type of a QSPI command descriptor.*

Macros

Transfer options

- #define `QSPI_CFG_CMD_MASK` (0xFFLU << 0LU)
- #define `QSPI_CFG_CMD(n)` ((n) << 0LU)
- #define `QSPI_CFG_CMD_MODE_MASK` (3LU << 8LU)
- #define `QSPI_CFG_CMD_MODE_NONE` (0LU << 8LU)
- #define `QSPI_CFG_CMD_MODE_ONE_LINE` (1LU << 8LU)
- #define `QSPI_CFG_CMD_MODE_TWO_LINES` (2LU << 8LU)
- #define `QSPI_CFG_CMD_MODE_FOUR_LINES` (3LU << 8LU)
- #define `QSPI_CFG_ADDR_MODE_MASK` (3LU << 10LU)

- #define **QSPI_CFG_ADDR_MODE_NONE** (0LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_ONE_LINE** (1LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_TWO_LINES** (2LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_FOUR_LINES** (3LU << 10LU)
- #define **QSPI_CFG_ADDR_SIZE_MASK** (3LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_8** (0LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_16** (1LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_24** (2LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_32** (3LU << 12LU)
- #define **QSPI_CFG_ALT_MODE_MASK** (3LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_NONE** (0LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_ONE_LINE** (1LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_TWO_LINES** (2LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_FOUR_LINES** (3LU << 14LU)
- #define **QSPI_CFG_ALT_SIZE_MASK** (3LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_8** (0LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_16** (1LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_24** (2LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_32** (3LU << 16LU)
- #define **QSPI_CFG_DUMMY_CYCLES_MASK** (0x1FLU << 18LU)
- #define **QSPI_CFG_DUMMY_CYCLES(n)** ((n) << 18LU)
- #define **QSPI_CFG_DATA_MODE_MASK** (3LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_NONE** (0LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_ONE_LINE** (1LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_TWO_LINES** (2LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_FOUR_LINES** (3LU << 24LU)
- #define **QSPI_CFG_SIOO** (1LU << 28LU)
- #define **QSPI_CFG_DDRM** (1LU << 31LU)

QSPI configuration options

- #define **QSPI_USE_WAIT** TRUE
Enables synchronous APIs.
- #define **QSPI_USE_MUTUAL_EXCLUSION** TRUE
Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Macro Functions

- #define **qspiStartCommandl**(qspip, cmdp)
Sends a command without data phase.
- #define **qspiStartSendl**(qspip, cmdp, n, txbuf)
Sends data over the QSPI bus.
- #define **qspiStartReceiveI**(qspip, cmdp, n, rxbuf)
Receives data from the QSPI bus.
- #define **qspiMapFlashI**(qspip, cmdp, addrp) **qspi_lld_map_flash**(qspip, cmdp, addrp)
Maps in memory space a QSPI flash device.
- #define **qspiUnmapFlashI**(qspip) **qspi_lld_unmap_flash**(qspip)
Maps in memory space a QSPI flash device.

Low level driver helper macros

- #define **_qspi_wakeup_isr**(qspip)
Wakes up the waiting thread.
- #define **_qspi_isr_code**(qspip)
Common ISR code.

Enumerations

Functions

- void `qspiInit` (void)
QSPI Driver initialization.
- void `qspiObjectInit` (QSPIDriver *qspip)
Initializes the standard part of a `QSPIDriver` structure.
- void `qspiStart` (QSPIDriver *qspip, const QSPIConfig *config)
Configures and activates the QSPI peripheral.
- void `qspiStop` (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void `qspiStartCommand` (QSPIDriver *qspip, const qspi_command_t *cmdp)
Sends a command without data phase.
- void `qspiStartSend` (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void `qspiStartReceive` (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void `qspiCommand` (QSPIDriver *qspip, const qspi_command_t *cmdp)
Sends a command without data phase.
- void `qspiSend` (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void `qspiReceive` (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void `qspiMapFlash` (QSPIDriver *qspip, const qspi_command_t *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void `qspiUnmapFlash` (QSPIDriver *qspip)
Unmaps from memory space a QSPI flash device.
- void `qspiAcquireBus` (QSPIDriver *qspip)
Gains exclusive access to the QSPI bus.
- void `qspiReleaseBus` (QSPIDriver *qspip)
Releases exclusive access to the QSPI bus.

9.74.1 Detailed Description

QSPI Driver macros and structures.

9.75 hal_qspi_lld.c File Reference

PLATFORM QSPI subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `qspi_lld_init` (void)
Low level QSPI driver initialization.
- void `qspi_lld_start` (QSPIDriver *qspip)
Configures and activates the QSPI peripheral.

- void [qspi_lld_stop](#) (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void [qspi_lld_command](#) (QSPIDriver *qspip, const [qspi_command_t](#) *cmdp)
Sends a command without data phase.
- void [qspi_lld_send](#) (QSPIDriver *qspip, const [qspi_command_t](#) *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void [qspi_lld_receive](#) (QSPIDriver *qspip, const [qspi_command_t](#) *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void [qspi_lld_map_flash](#) (QSPIDriver *qspip, const [qspi_command_t](#) *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void [qspi_lld_unmap_flash](#) (QSPIDriver *qspip)
Unmaps from memory space a QSPI flash device.

Variables

- [QSPIDriver QSPID1](#)
QSPID1 driver identifier.

9.75.1 Detailed Description

PLATFORM QSPI subsystem low level driver source.

9.76 hal_qspi_lld.h File Reference

PLATFORM QSPI subsystem low level driver header.

Data Structures

- struct [QSPIDriver](#)
Driver configuration structure.
- struct [QSPIDriver](#)
Structure representing an QSPI driver.

Macros

QSPI capabilities

- #define **QSPI_SUPPORTS_MEMMAP** TRUE

Configuration options

- #define [PLATFORM_QSPI_USE_QSPI1](#) FALSE
QSPID1 driver enable switch.

Typedefs

- typedef struct [QSPIDriver](#) [QSPIDriver](#)
Type of a structure representing an QSPI driver.
- typedef void(* [qspicallback_t](#)) ([QSPIDriver](#) *qspip)
Type of a QSPI notification callback.

Functions

- void `qspi_lld_init` (void)
Low level QSPI driver initialization.
- void `qspi_lld_start` (QSPIDriver *qspip)
Configures and activates the QSPI peripheral.
- void `qspi_lld_stop` (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void `qspi_lld_command` (QSPIDriver *qspip, const `qspi_command_t` *cmdp)
Sends a command without data phase.
- void `qspi_lld_send` (QSPIDriver *qspip, const `qspi_command_t` *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void `qspi_lld_receive` (QSPIDriver *qspip, const `qspi_command_t` *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void `qspi_lld_map_flash` (QSPIDriver *qspip, const `qspi_command_t` *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void `qspi_lld_unmap_flash` (QSPIDriver *qspip)
Unmaps from memory space a QSPI flash device.

9.76.1 Detailed Description

PLATFORM QSPI subsystem low level driver header.

9.77 hal_queues.c File Reference

I/O Queues code.

```
#include <string.h>
#include "hal.h"
```

Functions

- static size_t `iq_read` (input_queue_t *iqp, uint8_t *bp, size_t n)
Non-blocking input queue read.
- static size_t `oq_write` (output_queue_t *oqp, const uint8_t *bp, size_t n)
Non-blocking output queue write.
- void `iqObjectInit` (input_queue_t *iqp, uint8_t *bp, size_t size, `qnotify_t` infy, void *link)
Initializes an input queue.
- void `iqResetI` (input_queue_t *iqp)
Resets an input queue.
- msg_t `iqPutI` (input_queue_t *iqp, uint8_t b)
Input queue write.
- msg_t `iqGetI` (input_queue_t *iqp)
Input queue non-blocking read.
- msg_t `iqGetTimeout` (input_queue_t *iqp, `sysinterval_t` timeout)
Input queue read with timeout.
- size_t `iqReadI` (input_queue_t *iqp, uint8_t *bp, size_t n)
Input queue non-blocking read.
- size_t `iqReadTimeout` (input_queue_t *iqp, uint8_t *bp, size_t n, `sysinterval_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 [oqResetI](#) ([output_queue_t](#) *oqp)
Resets an output queue.
- [msg_t](#) [oqPutI](#) ([output_queue_t](#) *oqp, uint8_t b)
Output queue non-blocking write.
- [msg_t](#) [oqPutTimeout](#) ([output_queue_t](#) *oqp, uint8_t b, [sysinterval_t](#) timeout)
Output queue write with timeout.
- [msg_t](#) [oqGetI](#) ([output_queue_t](#) *oqp)
Output queue read.
- size_t [oqWriteI](#) ([output_queue_t](#) *oqp, const uint8_t *bp, size_t n)
Output queue non-blocking write.
- size_t [oqWriteTimeout](#) ([output_queue_t](#) *oqp, const uint8_t *bp, size_t n, [sysinterval_t](#) timeout)
Output queue write with timeout.

9.77.1 Detailed Description

I/O Queues code.

9.78 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_TIMEOUT
Queue empty.
- #define [Q_FULL](#) MSG_TIMEOUT
Queue full,.

Macro Functions

- #define [qSizeX](#)(qp)
Returns the queue's buffer size.
- #define [qSpacel](#)(qp) ((qp)->q_counter)
Queue space.
- #define [qGetLink](#)(qp) ((qp)->q_link)
Returns the queue application-defined link.

- `#define iqGetFullI(iqp) qSpacel(iqp)`
Returns the filled space into an input queue.
- `#define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpacel(iqp))`
Returns the empty space into an input queue.
- `#define iqIsEmptyI(iqp) ((bool)(qSpacel(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) - qSpacel(oqp))`
Returns the filled space into an output queue.
- `#define oqGetEmptyI(oqp) qSpacel(oqp)`
Returns the empty space into an output queue.
- `#define oqIsEmptyI(oqp)`
Evaluates to `true` if the specified output queue is empty.
- `#define oqIsFullI(oqp) ((bool)(qSpacel(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 iqResetI (input_queue_t *iqp)`
Resets an input queue.
- `msg_t iqPutI (input_queue_t *iqp, uint8_t b)`
Input queue write.
- `msg_t iqGetI (input_queue_t *iqp)`
Input queue non-blocking read.
- `msg_t iqGetTimeout (input_queue_t *iqp, sysinterval_t timeout)`
Input queue read with timeout.
- `size_t iqReadI (input_queue_t *iqp, uint8_t *bp, size_t n)`
Input queue non-blocking read.
- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, sysinterval_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 oqResetI (output_queue_t *oqp)`
Resets an output queue.
- `msg_t oqPutI (output_queue_t *oqp, uint8_t b)`

Output queue non-blocking write.

- `msg_t oqPutTimeout` (`output_queue_t *oqp`, `uint8_t b`, `sysinterval_t timeout`)

Output queue write with timeout.

- `msg_t oqGetl` (`output_queue_t *oqp`)

Output queue read.

- `size_t oqWritel` (`output_queue_t *oqp`, `const uint8_t *bp`, `size_t n`)

Output queue non-blocking write.

- `size_t oqWriteTimeout` (`output_queue_t *oqp`, `const uint8_t *bp`, `size_t n`, `sysinterval_t timeout`)

Output queue write with timeout.

9.78.1 Detailed Description

I/O Queues macros and structures.

9.79 hal_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 RTCAalarm *alarmspec`)
Set alarm time.
- void `rtcGetAlarm` (`RTCDriver *rtcp`, `rtcalarm_t alarm`, `RTCAalarm *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 `rtcConvertStructTmToDateTime` (`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.

9.79.1 Detailed Description

RTC Driver code.

9.80 hal_rtc.h File Reference

RTC Driver macros and structures.

```
#include <time.h>
#include "hal_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](#) 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

Typedefs

- typedef struct [RTCDriver](#) [RTCDriver](#)
Type of a structure representing an RTC driver.

Functions

- void [rtclnit](#) (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)

- void [rtcConvertDateTimeToStructTm](#) (const [RTCDatetime](#) *timespec, struct tm *timp, uint32_t *tv_msec)
Convert [RTCDatetime](#) to broken-down time structure.
- void [rtcConvertStructTmToDateTime](#) (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.

9.80.1 Detailed Description

RTC Driver macros and structures.

9.81 hal_rtc_lld.c File Reference

PLATFORM RTC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [rtc_lld_init](#) (void)
RTC driver identifier.
- 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 [RTCArm](#) *alarmspec)
Set alarm time.
- void [rtc_lld_get_alarm](#) ([RTCDriver](#) *rtcp, [rtcalarm_t](#) alarm, [RTCArm](#) *alarmspec)
Get alarm time.

9.81.1 Detailed Description

PLATFORM RTC subsystem low level driver source.

9.82 hal_rtc_lld.h File Reference

PLATFORM RTC subsystem low level driver header.

Data Structures

- struct [RTCArm](#)
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 in 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

Functions

- void `rtc_lld_init` (void)
RTC driver identifier.
- 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 `RTCArm` *alarmspec)
Set alarm time.
- void `rtc_lld_get_alarm` (`RTCDriver` *rtcp, `rtcalarm_t` alarm, `RTCArm` *alarmspec)
Get alarm time.

9.82.1 Detailed Description

PLATFORM RTC subsystem low level driver header.

9.83 hal_sdc.c File Reference

SDC Driver code.

```
#include <string.h>
#include "hal.h"
```


Enumerations

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 [sdclnit](#) (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.

Variables

- static const struct [SDCDriverVMT](#) [sdc_vmt](#)
Virtual methods table.

9.83.1 Detailed Description

SDC Driver code.

9.84 hal_sdc.h File Reference

SDC Driver macros and structures.

```
#include "hal_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.
- #define [SDC_INIT_OCR_V20](#) 0x50FF8000U
OCR initialization constant for V20 cards.
- #define [SDC_INIT_OCR](#) 0x80100000U
OCR initialization constant for non-V20 cards.

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

Functions

- void `sdclnit` (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.

9.84.1 Detailed Description

SDC Driver macros and structures.

9.85 hal_sdc_ild.c File Reference

PLATFORM SDC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `sdc_ild_init` (void)
Low level SDC driver initialization.
- void `sdc_ild_start` (SDCDriver *sdcp)
Configures and activates the SDC peripheral.
- void `sdc_ild_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.

9.85.1 Detailed Description

PLATFORM SDC subsystem low level driver source.

9.86 hal_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_mmc_sdc_block_device_methods`
SDC Driver 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 `SDC_Driver` `SDC_Driver`
Type of a structure representing an SDC driver.

Functions

- void `sdc_ild_init` (void)
Low level SDC driver initialization.
- void `sdc_ild_start` (`SDC_Driver` *sdc)
Configures and activates the SDC peripheral.
- void `sdc_ild_stop` (`SDC_Driver` *sdc)
Deactivates the SDC peripheral.
- void `sdc_ild_start_clk` (`SDC_Driver` *sdc)
Starts the SDIO clock and sets it to init mode (400kHz or less).
- void `sdc_ild_set_data_clk` (`SDC_Driver` *sdc, `sdcbusclk_t` clk)
Sets the SDIO clock to data mode (25MHz or less).
- void `sdc_ild_stop_clk` (`SDC_Driver` *sdc)
Stops the SDIO clock.
- void `sdc_ild_set_bus_mode` (`SDC_Driver` *sdc, `sdcbusmode_t` mode)
Switches the bus to 4 bits mode.
- void `sdc_ild_send_cmd_none` (`SDC_Driver` *sdc, uint8_t cmd, uint32_t arg)
Sends an SDIO command with no response expected.
- bool `sdc_ild_send_cmd_short` (`SDC_Driver` *sdc, uint8_t cmd, uint32_t arg, uint32_t *resp)
Sends an SDIO command with a short response expected.
- bool `sdc_ild_send_cmd_short_crc` (`SDC_Driver` *sdc, uint8_t cmd, uint32_t arg, uint32_t *resp)
Sends an SDIO command with a short response expected and CRC.
- bool `sdc_ild_send_cmd_long_crc` (`SDC_Driver` *sdc, uint8_t cmd, uint32_t arg, uint32_t *resp)
Sends an SDIO command with a long response expected and CRC.
- bool `sdc_ild_read` (`SDC_Driver` *sdc, uint32_t startblk, uint8_t *buf, uint32_t n)
Reads one or more blocks.
- bool `sdc_ild_write` (`SDC_Driver` *sdc, uint32_t startblk, const uint8_t *buf, uint32_t n)
Writes one or more blocks.
- bool `sdc_ild_sync` (`SDC_Driver` *sdc)
Waits for card idle condition.

9.86.1 Detailed Description

PLATFORM SDC subsystem low level driver header.

9.87 hal_sensors.h File Reference

Generic sensors interface header.

Data Structures

- struct [BaseSensorVMT](#)
BaseSensor virtual methods table.
- struct [BaseSensor](#)
Base stream class.

Macros

- #define [_base_sensor_methods_alone](#)
BaseSensor specific methods.
- #define [_base_sensor_methods](#)
BaseSensor specific methods with inherited ones.
- #define [_base_sensor_data](#)
BaseSensor specific data.

Macro Functions (BaseSensor)

- #define [sensorGetChannelNumber](#)(ip) (ip)->vmt->get_channels_number(ip)
Sensors get channels number.
- #define [sensorReadRaw](#)(ip, dp) (ip)->vmt->read_raw(ip, dp)
Sensors read raw data.
- #define [sensorReadCooked](#)(ip, dp) (ip)->vmt->read_cooked(ip, dp)
Sensors read cooked data.

9.87.1 Detailed Description

Generic sensors interface header.

9.88 hal_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 `sdIncomingDataI` (`SerialDriver` *sdp, uint8_t b)
Handles incoming data.
- `msg_t` `sdRequestDataI` (`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`.
- `msg_t` `sdControl` (`SerialDriver` *sdp, unsigned int operation, void *arg)
Control operation on a serial port.

9.88.1 Detailed Description

Serial Driver code.

9.89 hal_serial.h File Reference

Serial Driver macros and structures.

```
#include "hal_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.
- `#define` `SD_QUEUE_FULL_ERROR` (`eventflags_t`)1024
Queue full.

Serial configuration options

- #define `SERIAL_DEFAULT_BITRATE` 38400
Default bit rate.
- #define `SERIAL_BUFFERS_SIZE` 16
Serial buffers size.

Macro Functions

- #define `sdPutl(sdp, b) oqPutl(&(sdp)->queue, b)`
Direct write to a `SerialDriver`.
- #define `sdPut(sdp, b) oqPut(&(sdp)->queue, b)`
Direct write to a `SerialDriver`.
- #define `sdPutTimeout(sdp, b, t) oqPutTimeout(&(sdp)->queue, b, t)`
Direct write to a `SerialDriver` with timeout specification.
- #define `sdGetl(sdp) iqGetl(&(sdp)->iqueue)`
Direct read from a `SerialDriver`.
- #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 `sdWritel(sdp, b, n) oqWritel(&(sdp)->queue, b, n)`
Direct blocking write to a `SerialDriver`.
- #define `sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->queue, b, n, TIME_INFINITE)`
Direct blocking write to a `SerialDriver`.
- #define `sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->queue, b, n, t)`
Direct blocking write to a `SerialDriver` with timeout specification.
- #define `sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->queue, b, n, TIME_IMMEDIATE)`
Direct non-blocking write to a `SerialDriver`.
- #define `sdReadl(sdp, b, n) iqReadl(&(sdp)->iqueue, b, n, TIME_INFINITE)`
Direct blocking read from 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

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](#).
- [msg_t sdControl](#) ([SerialDriver](#) *sdp, unsigned int operation, void *arg)
Control operation on a serial port.

9.89.1 Detailed Description

Serial Driver macros and structures.

9.90 hal_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.

9.90.1 Detailed Description

PLATFORM serial subsystem low level driver source.

9.91 hal_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.

9.91.1 Detailed Description

PLATFORM serial subsystem low level driver header.

9.92 hal_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 [sdulnit](#) (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 [sduSuspendHookI](#) ([SerialUSBDriver](#) *sdup)

- *USB device suspend handler.*
- void [sduWakeupHookI](#) ([SerialUSBDriver](#) *sdup)
- *USB device wakeup 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.*
- [msg_t sduControl](#) ([USBDriver](#) *usbp, unsigned int operation, void *arg)
- *Control operation on a serial USB port.*

9.92.1 Detailed Description

Serial over USB Driver code.

9.93 hal_serial_usb.h File Reference

Serial over USB Driver macros and structures.

```
#include "hal_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

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 [sduSuspendHookI](#) ([SerialUSBDriver](#) *sdup)
USB device suspend handler.
- void [sduWakeupHookI](#) ([SerialUSBDriver](#) *sdup)
USB device wakeup 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.
- [msg_t sduControl](#) ([USBDriver](#) *usbp, unsigned int operation, void *arg)
Control operation on a serial USB port.

9.93.1 Detailed Description

Serial over USB Driver macros and structures.

9.94 hal_spi.c File Reference

SPI Driver code.

```
#include "hal.h"
```

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 `spiAbortI` (SPIDriver *spip)
Aborts the ongoing SPI operation.
- void `spiAbort` (SPIDriver *spip)
Aborts the ongoing SPI operation, if any.
- void `spiIgnore` (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.

9.94.1 Detailed Description

SPI Driver code.

9.95 hal_spi.h File Reference

SPI Driver macros and structures.

```
#include "hal_spi_llld.h"
```

Macros

Chip Select modes

- `#define SPI_SELECT_MODE_NONE`
- `#define SPI_SELECT_MODE_PAD 1` */** @brief Legacy mode. */*
- `#define SPI_SELECT_MODE_PORT 2` */** @brief Fastest mode. */*
- `#define SPI_SELECT_MODE_LINE 3` */** @brief Packed mode. */*
- `#define SPI_SELECT_MODE_LLD 4` */** @brief LLD-defined mode.*/*

SPI configuration options

- `#define SPI_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define SPI_USE_CIRCULAR FALSE`
Enables circular transfers APIs.
- `#define SPI_USE_MUTUAL_EXCLUSION TRUE`
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.
- `#define SPI_SELECT_MODE SPI_SELECT_MODE_PAD`
Handling method for SPI CS line.

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 spiStartReceive(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.
- `#define _spi_isr_code_half1(spip)`
Common ISR code in circular mode.
- `#define _spi_isr_code_half2(spip)`
Common ISR code in circular mode.

Enumerations

Functions

- void `spiInit` (void)
SPI Driver initialization.
- void `spiObjectInit` (SPIDriver *spip)
Initializes the standard part of a `SPIDriver` structure.

- void `spiStart` (`SPIDriver *spip`, const `SPIDriverConfig *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 `spiAbortl` (`SPIDriver *spip`)
Aborts the ongoing SPI operation.
- void `spiAbort` (`SPIDriver *spip`)
Aborts the ongoing SPI operation, if any.
- void `spiIgnore` (`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.

9.95.1 Detailed Description

SPI Driver macros and structures.

9.96 hal_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 *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.
- void [spi_lld_abort](#) ([SPIDriver](#) *spip)
Aborts the ongoing SPI operation, if any.
- uint16_t [spi_lld_polled_exchange](#) ([SPIDriver](#) *spip, uint16_t frame)
Exchanges one frame using a polled wait.

Variables

- [SPIDriver SPI1](#)
SPI1 driver identifier.

9.96.1 Detailed Description

PLATFORM SPI subsystem low level driver source.

9.97 hal_spi_lld.h File Reference

PLATFORM SPI subsystem low level driver header.

Data Structures

- struct [SPICongfig](#)
Driver configuration structure.
- struct [SPIDriver](#)
Structure representing an SPI driver.

Macros

- #define [SPI_SUPPORTS_CIRCULAR](#) TRUE
Circular mode support flag.

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.
- void [spi_lld_abort](#) ([SPIDriver](#) *spip)
Aborts the ongoing SPI operation, if any.
- uint16_t [spi_lld_polled_exchange](#) ([SPIDriver](#) *spip, uint16_t frame)
Exchanges one frame using a polled wait.

9.97.1 Detailed Description

PLATFORM SPI subsystem low level driver header.

9.98 hal_st.c File Reference

ST Driver code.

```
#include "hal.h"
```

Functions

- void [stInit](#) (void)
ST Driver initialization.
- void [stStartAlarm](#) (sys_time_t abstime)
Starts the alarm.
- void [stStopAlarm](#) (void)

Stops the alarm interrupt.

- void `stSetAlarm` (`sys_time_t` abstime)

Sets the alarm time.

- `sys_time_t` `stGetAlarm` (void)

Returns the current alarm time.

9.98.1 Detailed Description

ST Driver code.

9.99 hal_st.h File Reference

ST Driver macros and structures.

```
#include "hal_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` (`sys_time_t` abstime)
Starts the alarm.
- void `stStopAlarm` (void)
Stops the alarm interrupt.
- void `stSetAlarm` (`sys_time_t` abstime)
Sets the alarm time.
- `sys_time_t` `stGetAlarm` (void)
Returns the current alarm time.

9.99.1 Detailed Description

ST Driver macros and structures.

This header is designed to be include-able without having to include other files from the HAL.

9.100 hal_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.

9.100.1 Detailed Description

PLATFORM ST subsystem low level driver source.

9.101 hal_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.

9.101.1 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.

9.102 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_base_object_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.

9.102.1 Detailed Description

Data streams.

This header defines abstract interfaces useful to access generic data streams in a standardized way.

9.103 hal_thermometer.h File Reference

Generic thermometer interface header.

```
#include "hal_sensors.h"
```

Data Structures

- struct `BaseThermometerVMT`
BaseThermometer virtual methods table.
- struct `BaseThermometer`
Base thermometer class.

Macros

- #define `_base_thermometer_methods_alone`
BaseThermometer specific methods.
- #define `_base_thermometer_methods`
BaseThermometer specific methods with inherited ones.
- #define `_base_thermometer_data_base_sensor_data`
BaseThermometer specific data.

Macro Functions (BaseThermometer)

- #define `thermometerGetChannelsNumber(ip)` (ip)->vmt->get_channels_number(ip)
Thermometer get channels number.
- #define `thermometerReadRaw(ip, dp)` (ip)->vmt->read_raw(ip, dp)
Thermometer read raw data.
- #define `thermometerReadCooked(ip, dp)` (ip)->vmt->read_cooked(ip, dp)
Thermometer read cooked data.
- #define `thermometerSetBias(ip, bp)` (ip)->vmt->set_bias(ip, bp)
Updates thermometer bias data from received buffer.
- #define `thermometerResetBias(ip)` (ip)->vmt->reset_bias(ip)
Reset thermometer bias data restoring it to zero.
- #define `thermometerSetSensitivity(ip, sp)` (ip)->vmt->set_sensitivity(ip, sp)
Updates thermometer sensitivity data from received buffer.
- #define `thermometerResetSensitivity(ip)` (ip)->vmt->reset_sensitivity(ip)
Reset thermometer sensitivity data restoring it to its typical value.

9.103.1 Detailed Description

Generic thermometer interface header.

9.104 hal_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 `uartStartSendI` (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 `uartStopSendI` (UARTDriver *uartp)
Stops any ongoing transmission.
- void `uartStartReceive` (UARTDriver *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- void `uartStartReceiveI` (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 uartStopReceive (UARTDriver *uartp)`
Stops any ongoing receive operation.
- `msg_t uartSendTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)`
Performs a transmission on the UART peripheral.
- `msg_t uartSendFullTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)`
Performs a transmission on the UART peripheral.
- `msg_t uartReceiveTimeout (UARTDriver *uartp, size_t *np, void *rxbuf, sysinterval_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.

9.104.1 Detailed Description

UART Driver code.

9.105 hal_uart.h File Reference

UART Driver macros and structures.

```
#include "hal_uart_lld.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`
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 error conditions

- `#define UART_ERR_NOT_ACTIVE (size_t)-1`

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_wakeup_rx_timeout_isr(uartp)`
Wakes up the waiting thread in case of RX timeout.
- `#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.
- `#define _uart_timeout_isr_code(uartp)`
Timeout ISR code for receiver.

Enumerations

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 `uartStartSendI` (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 `uartStopSendI` (UARTDriver *uartp)
Stops any ongoing transmission.
- void `uartStartReceive` (UARTDriver *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- void `uartStartReceiveI` (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 `uartStopReceiveI` (UARTDriver *uartp)
Stops any ongoing receive operation.
- msg_t `uartSendTimeout` (UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)
Performs a transmission on the UART peripheral.

- `msg_t uartSendFullTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)`
Performs a transmission on the UART peripheral.
- `msg_t uartReceiveTimeout (UARTDriver *uartp, size_t *np, void *rxbuf, sysinterval_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.

9.105.1 Detailed Description

UART Driver macros and structures.

9.106 hal_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_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.

Variables

- `UARTDriver UARTD1`
UART1 driver identifier.

9.106.1 Detailed Description

PLATFORM UART subsystem low level driver source.

9.107 hal_uart_lld.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(* [uartercb_t](#)) ([UARTDriver](#) *uartp, [uartflags_t](#) e)
Receive error UART notification callback type.

Functions

- void [uart_ild_init](#) (void)
Low level UART driver initialization.
- void [uart_ild_start](#) ([UARTDriver](#) *uartp)
Configures and activates the UART peripheral.
- void [uart_ild_stop](#) ([UARTDriver](#) *uartp)
Deactivates the UART peripheral.
- void [uart_ild_start_send](#) ([UARTDriver](#) *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t [uart_ild_stop_send](#) ([UARTDriver](#) *uartp)
Stops any ongoing transmission.
- void [uart_ild_start_receive](#) ([UARTDriver](#) *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t [uart_ild_stop_receive](#) ([UARTDriver](#) *uartp)
Stops any ongoing receive operation.

9.107.1 Detailed Description

PLATFORM UART subsystem low level driver header.

9.108 hal_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)
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 [usbStallReceiveI](#) (USBDriver *usbp, usbep_t ep)
Stalls an OUT endpoint.
- bool [usbStallTransmitI](#) (USBDriver *usbp, usbep_t ep)
Stalls an IN endpoint.
- void [usbWakeupHost](#) (USBDriver *usbp)
Host wake-up procedure.
- 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.

9.108.1 Detailed Description

USB Driver code.

9.109 hal_usb.h File Reference

USB Driver macros and structures.

```
#include "hal_usb_lld.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, b↵
MaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNum↵
Configurations)
Device Descriptor helper macro.
- #define [USB_DESC_CONFIGURATION_SIZE](#) 9U
Configuration Descriptor size.
- #define [USB_DESC_CONFIGURATION](#)(wTotalLength, bNumInterfaces, bConfigurationValue, i↵
Configuration, bmAttributes, bMaxPower)
Configuration Descriptor helper macro.
- #define [USB_DESC_INTERFACE_SIZE](#) 9U
Interface Descriptor size.
- #define [USB_DESC_INTERFACE](#)(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterface↵
Class, 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, b↵
FunctionSubClass, bFunctionProcotol, 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 `usbGetDriverStatel(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 `usbGetTransmitStatusl(usbp, ep)` (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)
Returns the status of an IN endpoint.
- #define `usbGetReceiveStatusl(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

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 [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)
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 [usbStallReceiveI](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Stalls an OUT endpoint.
- bool [usbStallTransmitI](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Stalls an IN endpoint.
- void [usbWakeupHost](#) ([USBDriver](#) *usbp)
Host wake-up procedure.
- 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.

9.109.1 Detailed Description

USB Driver macros and structures.

9.110 hal_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

9.110.1 Detailed Description

USB CDC macros and structures.

9.111 hal_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.

9.111.1 Detailed Description

PLATFORM USB subsystem low level driver source.

9.111.2 Variable Documentation

9.111.2.1 [USBInEndpointState in](#)

IN EP0 state.

9.111.2.2 [USBOutEndpointState out](#)

OUT EP0 state.

9.112 [hal_usb_ild.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_llid_get_frame_number](#)(usbp) 0
Returns the current frame number.
- #define [usb_llid_get_transaction_size](#)(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)
Returns the exact size of a receive transaction.
- #define [usb_llid_connect_bus](#)(usbp)
Connects the USB device.
- #define [usb_llid_disconnect_bus](#)(usbp)
Disconnect the USB device.
- #define [usb_llid_wakeup_host](#)(usbp)
Start of host wake-up procedure.

PLATFORM configuration options

- #define [PLATFORM_USB_USE_USB1](#) FALSE
USB driver enable switch.

Functions

- void [usb_llid_init](#) (void)
Low level USB driver initialization.
- void [usb_llid_start](#) (USBDriver *usbp)
Configures and activates the USB peripheral.
- void [usb_llid_stop](#) (USBDriver *usbp)
Deactivates the USB peripheral.
- void [usb_llid_reset](#) (USBDriver *usbp)
USB low level reset routine.
- void [usb_llid_set_address](#) (USBDriver *usbp)
Sets the USB address.
- void [usb_llid_init_endpoint](#) (USBDriver *usbp, [usbep_t](#) ep)
Enables an endpoint.
- void [usb_llid_disable_endpoints](#) (USBDriver *usbp)
Disables all the active endpoints except the endpoint zero.
- [usbepstatus_t](#) [usb_llid_get_status_in](#) (USBDriver *usbp, [usbep_t](#) ep)
Returns the status of an IN endpoint.
- [usbepstatus_t](#) [usb_llid_get_status_out](#) (USBDriver *usbp, [usbep_t](#) ep)
Returns the status of an OUT endpoint.
- void [usb_llid_read_setup](#) (USBDriver *usbp, [usbep_t](#) ep, [uint8_t](#) *buf)
Reads a setup packet from the dedicated packet buffer.
- void [usb_llid_prepare_receive](#) (USBDriver *usbp, [usbep_t](#) ep)
Prepares for a receive operation.
- void [usb_llid_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.

9.112.1 Detailed Description

PLATFORM USB subsystem low level driver header.

9.113 `hal_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.

9.113.1 Detailed Description

WDG Driver code.

9.114 `hal_wdg.h` File Reference

WDG Driver macros and structures.

```
#include "hal_wdg_lld.h"
```

Macros

- #define `wdgResetI(wdgp) wdg_ild_reset(wdgp)`
Resets WDG's counter.

Enumerations

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.

9.114.1 Detailed Description

WDG Driver macros and structures.

9.115 hal_wdg_ild.c File Reference

WDG Driver subsystem low level driver source template.

```
#include "hal.h"
```

Functions

- void `wdg_ild_init` (void)
Low level WDG driver initialization.
- void `wdg_ild_start` (WDGDriver *wdgp)
Configures and activates the WDG peripheral.
- void `wdg_ild_stop` (WDGDriver *wdgp)
Deactivates the WDG peripheral.
- void `wdg_ild_reset` (WDGDriver *wdgp)
Reloads WDG's counter.

9.115.1 Detailed Description

WDG Driver subsystem low level driver source template.

9.116 hal_wdg_ild.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_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.

9.116.1 Detailed Description

WDG Driver subsystem low level driver header template.

9.117 halconf.h File Reference

HAL configuration header.

```
#include "mcuconf.h"
```

Macros

- #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_CRY](#) TRUE

- Enables the cryptographic subsystem.*

 - #define `HAL_USE_DAC` TRUE

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_QSPI` TRUE

Enables the QSPI 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

Enables the USB subsystem.
- #define `HAL_USE_WDG` TRUE

Enables the WDG subsystem.
- #define `PAL_USE_CALLBACKS` FALSE

Enables synchronous APIs.
- #define `PAL_USE_WAIT` FALSE

Enables synchronous APIs.
- #define `ADC_USE_WAIT` TRUE

Enables synchronous APIs.
- #define `ADC_USE_MUTUAL_EXCLUSION` TRUE

Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.
- #define `CAN_USE_SLEEP_MODE` TRUE

Sleep mode related APIs inclusion switch.
- #define `CAN_ENFORCE_USE_CALLBACKS` FALSE

Enforces the driver to use direct callbacks rather than OSAL events.
- #define `HAL_CRY_USE_FALLBACK` FALSE

Enables the SW fall-back of the cryptographic driver.

- #define `HAL_CRY_ENFORCE_FALLBACK` FALSE
Makes the driver forcibly use the fall-back implementations.
- #define `DAC_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `DAC_USE_MUTUAL_EXCLUSION` TRUE
Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.
- #define `I2C_USE_MUTUAL_EXCLUSION` TRUE
Enables the mutual exclusion APIs on the I2C bus.
- #define `MAC_USE_ZERO_COPY` TRUE
Enables the zero-copy API.
- #define `MAC_USE_EVENTS` TRUE
Enables an event sources for incoming packets.
- #define `MMC_NICE_WAITING` TRUE
Delays insertions.
- #define `QSPI_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `QSPI_USE_MUTUAL_EXCLUSION` TRUE
Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.
- #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.
- #define `SDC_INIT_OCR_V20` 0x50FF8000U
OCR initialization constant for V20 cards.
- #define `SDC_INIT_OCR` 0x80100000U
OCR initialization constant for non-V20 cards.
- #define `SERIAL_DEFAULT_BITRATE` 38400
Default bit rate.
- #define `SERIAL_BUFFERS_SIZE` 16
Serial buffers size.
- #define `SERIAL_USB_BUFFERS_SIZE` 256
Serial over USB buffers size.
- #define `SERIAL_USB_BUFFERS_NUMBER` 2
Serial over USB number of buffers.
- #define `SPI_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `SPI_USE_CIRCULAR` FALSE
Enables circular transfers APIs.
- #define `SPI_USE_MUTUAL_EXCLUSION` TRUE
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.
- #define `SPI_SELECT_MODE` SPI_SELECT_MODE_PAD
Handling method for SPI CS line.
- #define `UART_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `UART_USE_MUTUAL_EXCLUSION` TRUE
Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.
- #define `USB_USE_WAIT` TRUE
Enables synchronous APIs.

9.117.1 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.

9.118 mfs.c File Reference

Managed Flash Storage module code.

```
#include <string.h>
#include "hal.h"
#include "mfs.h"
```

Macros

- `#define RET_ON_ERROR(err)`
Error check helper.

Functions

- static `mfs_error_t mfs_flash_read` (`MFSDriver *mfsp`, `flash_offset_t` offset, `size_t` n, `uint8_t *rp`)
Flash read.
- static `mfs_error_t mfs_flash_write` (`MFSDriver *mfsp`, `flash_offset_t` offset, `size_t` n, `const uint8_t *wp`)
Flash write.
- static `mfs_error_t mfs_flash_copy` (`MFSDriver *mfsp`, `flash_offset_t` doffset, `flash_offset_t` soffset, `uint32_t` n)
Flash copy.
- static `mfs_error_t mfs_record_check` (`MFSDriver *mfsp`, `mfs_data_header_t *dhdrp`, `flash_offset_t` offset, `flash_offset_t` limit, `mfs_record_state_t *sts`)
Verifies integrity of a record.
- static `mfs_error_t mfs_bank_erase` (`MFSDriver *mfsp`, `mfs_bank_t` bank)
Erases and verifies all sectors belonging to a bank.
- static `mfs_error_t mfs_bank_verify_erase` (`MFSDriver *mfsp`, `mfs_bank_t` bank)
Erases and verifies all sectors belonging to a bank.
- static `mfs_error_t mfs_bank_write_header` (`MFSDriver *mfsp`, `mfs_bank_t` bank, `uint32_t` cnt)
Writes the validation header in a bank.
- static `mfs_error_t mfs_bank_scan_records` (`MFSDriver *mfsp`, `mfs_bank_t` bank, `mfs_bank_state_t *statep`)
Scans blocks searching for records.
- static `mfs_error_t mfs_bank_get_state` (`MFSDriver *mfsp`, `mfs_bank_t` bank, `mfs_bank_state_t *statep`, `uint32_t *cntp`)
Determines the state of a bank.
- static `mfs_error_t mfs_bank_mount` (`MFSDriver *mfsp`, `mfs_bank_t` bank, `mfs_bank_state_t *statep`)
Selects a bank as current.
- static `mfs_error_t mfs_garbage_collect` (`MFSDriver *mfsp`)
Enforces a garbage collection.
- static `mfs_error_t mfs_try_mount` (`MFSDriver *mfsp`)
Performs a flash partition mount attempt.
- `mfs_error_t mfs_mount` (`MFSDriver *mfsp`)
Configures and activates a MFS driver.
- void `mfsObjectInit` (`MFSDriver *mfsp`)

- Initializes an instance.*
- `mfs_error_t mfsStart (MFSDriver *mfsp, const MFSCConfig *config)`
Configures and activates a MFS driver.
- `void mfsStop (MFSDriver *mfsp)`
Deactivates a MFS driver.
- `mfs_error_t mfsErase (MFSDriver *mfsp)`
Destroys the state of the managed storage by erasing the flash.
- `mfs_error_t mfsReadRecord (MFSDriver *mfsp, mfs_id_t id, size_t *np, uint8_t *buffer)`
Retrieves and reads a data record.
- `mfs_error_t mfsWriteRecord (MFSDriver *mfsp, mfs_id_t id, size_t n, const uint8_t *buffer)`
Creates or updates a data record.
- `mfs_error_t mfsEraseRecord (MFSDriver *mfsp, mfs_id_t id)`
Erases a data record.
- `mfs_error_t mfsPerformGarbageCollection (MFSDriver *mfsp)`
Enforces a garbage collection operation.

9.118.1 Detailed Description

Managed Flash Storage module code.

This module manages a flash partition as a generic storage where arbitrary data records can be created, updated, deleted and retrieved.

A managed partition is composed of two banks of equal size, a bank is composed of one or more erasable sectors, a sector is divided in writable pages.

The module handles flash wear leveling and recovery of damaged banks (where possible) caused by power loss during operations. Both operations are transparent to the user.

9.119 mfs.h File Reference

Managed Flash Storage module header.

```
#include "hal_flash.h"
```

Data Structures

- union `mfs_bank_header_t`
Type of a bank header.
- union `mfs_data_header_t`
Type of a data block header.
- struct `MFSCConfig`
Type of a MFS configuration structure.
- struct `MFSDriver`
Type of an MFS instance.

Macros

Configuration options

- `#define MFS_CFG_MAX_RECORDS 32`
Maximum number of indexed records in the managed storage.
- `#define MFS_CFG_MAX_REPAIR_ATTEMPTS 3`

- *Maximum number of repair attempts on partition mount.*
• #define `MFS_CFG_WRITE_VERIFY` TRUE
- *Verify written data.*
• #define `MFS_CFG_STRONG_CHECKING` TRUE
- *Enables a stronger and slower check procedure on mount.*
• #define `MFS_CFG_BUFFER_SIZE` 32
- *Size of the buffer used for data copying.*

Error codes handling macros

- #define `MFS_IS_ERROR`(err) ((err) < MFS_NO_ERROR)
- #define `MFS_IS_WARNING`(err) ((err) > MFS_NO_ERROR)

Typedefs

- typedef uint32_t `mfs_id_t`
Type of a record identifier.

Enumerations

Functions

- void `mfsObjectInit` (MFSDriver *mfsp)
Initializes an instance.
- `mfs_error_t` `mfsStart` (MFSDriver *mfsp, const MFSCfg *config)
Configures and activates a MFS driver.
- void `mfsStop` (MFSDriver *mfsp)
Deactivates a MFS driver.
- `mfs_error_t` `mfsErase` (MFSDriver *mfsp)
Destroys the state of the managed storage by erasing the flash.
- `mfs_error_t` `mfsReadRecord` (MFSDriver *mfsp, `mfs_id_t` id, size_t *np, uint8_t *buffer)
Retrieves and reads a data record.
- `mfs_error_t` `mfsWriteRecord` (MFSDriver *mfsp, `mfs_id_t` id, size_t n, const uint8_t *buffer)
Creates or updates a data record.
- `mfs_error_t` `mfsEraseRecord` (MFSDriver *mfsp, `mfs_id_t` id)
Erases a data record.
- `mfs_error_t` `mfsPerformGarbageCollection` (MFSDriver *mfsp)
Enforces a garbage collection operation.

9.119.1 Detailed Description

Managed Flash Storage module header.

9.120 osal.c File Reference

OSAL module code.

```
#include "osal.h"
```

Functions

- void `osalInit` (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` (sysinterval_t time)
Suspends the invoking thread for the specified time.
- void `osalThreadSleep` (sysinterval_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, sysinterval_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, sysinterval_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.

Variables

- const char * `osal_halt_msg`
Pointer to a halt error message.

9.120.1 Detailed Description

OSAL module code.

9.121 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_TIMEOUT** ([msg_t](#))-1
- #define **MSG_RESET** ([msg_t](#))-2

Special time constants

- #define **TIME_IMMEDIATE** (([sysinterval_t](#))0)
- #define **TIME_INFINITE** (([sysinterval_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_S2I(secs)` ((`sysinterval_t`)(`uint32_t`(secs) * (`uint32_t`)`OSAL_ST_FREQUENCY`))
Seconds to system ticks.
- #define `OSAL_MS2I(msecs)`
Milliseconds to system ticks.
- #define `OSAL_US2I(usecs)`
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(secs)` `osalThreadSleep(OSAL_S2I(secs))`
Delays the invoking thread for the specified number of seconds.
- #define `osalThreadSleepMilliseconds(msecs)` `osalThreadSleep(OSAL_MS2I(msecs))`
Delays the invoking thread for the specified number of milliseconds.
- #define `osalThreadSleepMicroseconds(usecs)` `osalThreadSleep(OSAL_US2I(usecs))`
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 [sysinterval_t](#)
Type of system time interval.
- typedef uint32_t [rtcnt_t](#)
Type of realtime counter.
- typedef void * [thread_reference_t](#)
Type of a thread reference.
- typedef uint32_t [eventflags_t](#)
Type of an event flags mask.
- 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 [mutex_t](#)
Type of a mutex.

Functions

- void [osalInit](#) (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](#) ([sysinterval_t](#) time)
Suspends the invoking thread for the specified time.
- void [osalThreadSleep](#) ([sysinterval_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, [sysinterval_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, [sysinterval_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 [systime_t](#) [osalTimeAddX](#) ([systime_t](#) systime, [sysinterval_t](#) interval)
Adds an interval to a system time returning a system time.
- static [sysinterval_t](#) [osalTimeDiffX](#) ([systime_t](#) start, [systime_t](#) end)
Subtracts two system times returning an interval.
- static bool [osalTimeInRangeX](#) ([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 source object.
- static void [osalMutexObjectInit](#) ([mutex_t](#) *mp)
Initializes a `mutex_t` object.

Variables

- const char * [osal_halt_msg](#)
Pointer to a halt error message.

9.121.1 Detailed Description

OSAL module header.

Index

- [_CHIBIOS_HAL_](#)
 - [HAL Driver, 185](#)
- [_IOBUS_DATA](#)
 - [PAL Driver, 433](#)
- [_adc_isr_error_code](#)
 - [ADC Driver, 33](#)
- [_adc_isr_full_code](#)
 - [ADC Driver, 33](#)
- [_adc_isr_half_code](#)
 - [ADC Driver, 32](#)
- [_adc_reset_i](#)
 - [ADC Driver, 31](#)
- [_adc_reset_s](#)
 - [ADC Driver, 31](#)
- [_adc_timeout_isr](#)
 - [ADC Driver, 32](#)
- [_adc_wakeup_isr](#)
 - [ADC Driver, 32](#)
- [_base_accelerometer_data](#)
 - [Generic Accelerometer Interface, 24](#)
- [_base_accelerometer_methods](#)
 - [Generic Accelerometer Interface, 24](#)
- [_base_accelerometer_methods_alone](#)
 - [Generic Accelerometer Interface, 24](#)
- [_base_asynchronous_channel_data](#)
 - [Abstract I/O Channel, 216](#)
- [_base_asynchronous_channel_methods](#)
 - [Abstract I/O Channel, 216](#)
- [_base_barometer_data](#)
 - [Generic Barometer Interface, 46](#)
- [_base_barometer_methods](#)
 - [Generic Barometer Interface, 46](#)
- [_base_barometer_methods_alone](#)
 - [Generic Barometer Interface, 45](#)
- [_base_block_device_data](#)
 - [Abstract I/O Block Device, 226](#)
- [_base_block_device_methods](#)
 - [Abstract I/O Block Device, 226](#)
- [_base_channel_data](#)
 - [Abstract I/O Channel, 211](#)
- [_base_channel_methods](#)
 - [Abstract I/O Channel, 211](#)
- [_base_compass_data](#)
 - [Generic Compass Interface, 69](#)
- [_base_compass_methods](#)
 - [Generic Compass Interface, 69](#)
- [_base_compass_methods_alone](#)
 - [Generic Compass Interface, 68](#)
- [_base_flash_data](#)
 - [Generic Flash Driver Interface, 158](#)
- [_base_flash_methods](#)
 - [Generic Flash Driver Interface, 158](#)
- [_base_flash_methods_alone](#)
 - [Generic Flash Driver Interface, 158](#)
- [_base_gyroscope_data](#)
 - [Generic Gyroscope Interface, 180](#)
- [_base_gyroscope_methods](#)
 - [Generic Gyroscope Interface, 180](#)
- [_base_gyroscope_methods_alone](#)
 - [Generic Gyroscope Interface, 179](#)
- [_base_hygrometer_data](#)
 - [Generic Hygrometer Interface, 260](#)
- [_base_hygrometer_methods](#)
 - [Generic Hygrometer Interface, 260](#)
- [_base_hygrometer_methods_alone](#)
 - [Generic Hygrometer Interface, 259](#)
- [_base_object_data](#)
 - [Base Object, 232](#)
- [_base_object_methods](#)
 - [Base Object, 232](#)
- [_base_sensor_data](#)
 - [Generic Sensor Interface, 547](#)
- [_base_sensor_methods](#)
 - [Generic Sensor Interface, 546](#)
- [_base_sensor_methods_alone](#)
 - [Generic Sensor Interface, 546](#)
- [_base_sequential_stream_data](#)
 - [Abstract Streams, 257](#)
- [_base_sequential_stream_methods](#)
 - [Abstract Streams, 256](#)
- [_base_thermometer_data](#)
 - [Generic Thermometer Interface, 611](#)
- [_base_thermometer_methods](#)
 - [Generic Thermometer Interface, 611](#)
- [_base_thermometer_methods_alone](#)
 - [Generic Thermometer Interface, 610](#)
- [_can_error_isr](#)
 - [CAN Driver, 54](#)
- [_can_rx_full_isr](#)
 - [CAN Driver, 54](#)
- [_can_tx_empty_isr](#)
 - [CAN Driver, 54](#)
- [_can_wakeup_isr](#)
 - [CAN Driver, 54](#)
- [_dac_isr_error_code](#)
 - [DAC Driver, 132](#)
- [_dac_isr_full_code](#)
 - [DAC Driver, 131](#)

- `_dac_isr_half_code`
DAC Driver, [131](#)
- `_dac_reset_i`
DAC Driver, [130](#)
- `_dac_reset_s`
DAC Driver, [130](#)
- `_dac_timeout_isr`
DAC Driver, [131](#)
- `_dac_wait_s`
DAC Driver, [129](#)
- `_dac_wakeup_isr`
DAC Driver, [130](#)
- `_file_stream_data`
Abstract Files, [220](#)
- `_file_stream_methods`
Abstract Files, [220](#)
- `_i2c_wakeup_error_isr`
I2C Driver, [267](#)
- `_i2c_wakeup_isr`
I2C Driver, [267](#)
- `_i2s_isr_full_code`
I2S Driver, [280](#)
- `_i2s_isr_half_code`
I2S Driver, [280](#)
- `_icu_isr_invoke_overflow_cb`
ICU Driver, [293](#)
- `_icu_isr_invoke_period_cb`
ICU Driver, [293](#)
- `_icu_isr_invoke_width_cb`
ICU Driver, [293](#)
- `_jesd216_flash_data`
JESD216 Compliant Flash Driver Interface, [306](#)
- `_jesd216_flash_methods`
JESD216 Compliant Flash Driver Interface, [306](#)
- `_jesd216_flash_methods_alone`
JESD216 Compliant Flash Driver Interface, [306](#)
- `_mmc_driver_methods`
MMC over SPI Driver, [395](#)
- `_mmcscd_block_device_data`
MMC/SD Block Device, [419](#)
- `_mmcscd_block_device_methods`
MMC/SD Block Device, [418](#)
- `_mmcscd_get_capacity`
MMC/SD Block Device, [420](#)
- `_mmcscd_get_capacity_ext`
MMC/SD Block Device, [421](#)
- `_mmcscd_get_slice`
MMC/SD Block Device, [420](#)
- `_mmcscd_unpack_csd_mmc`
MMC/SD Block Device, [422](#)
- `_mmcscd_unpack_csd_v10`
MMC/SD Block Device, [423](#)
- `_mmcscd_unpack_csd_v20`
MMC/SD Block Device, [423](#)
- `_mmcscd_unpack_mmc_cid`
MMC/SD Block Device, [422](#)
- `_mmcscd_unpack_sdc_cid`
MMC/SD Block Device, [421](#)
- `_pal_clear_event`
PAL Driver, [434](#), [435](#)
- `_pal_init_event`
PAL Driver, [433](#), [434](#)
- `_pal_isr_code`
PAL Driver, [436](#)
- `_pal_lld_init`
PAL Driver, [463](#)
- `_pal_lld_setgroupmode`
PAL Driver, [463](#)
- `_qspi_isr_code`
QSPI Driver, [495](#)
- `_qspi_wakeup_isr`
QSPI Driver, [494](#)
- `_rtc_driver_methods`
RTC Driver, [510](#)
- `_sdcard_driver_methods`
SDC Driver, [523](#)
- `_sdcard_wait_for_transfer_state`
SDC Driver, [531](#)
- `_serial_driver_data`
Serial Driver, [556](#)
- `_serial_driver_methods`
Serial Driver, [552](#)
- `_serial_usb_driver_data`
Serial over USB Driver, [567](#)
- `_serial_usb_driver_methods`
Serial over USB Driver, [567](#)
- `_spi_isr_code`
SPI Driver, [584](#)
- `_spi_isr_code_half1`
SPI Driver, [584](#)
- `_spi_isr_code_half2`
SPI Driver, [585](#)
- `_spi_wakeup_isr`
SPI Driver, [583](#)
- `_uart_rx_complete_isr_code`
UART Driver, [624](#)
- `_uart_rx_error_isr_code`
UART Driver, [625](#)
- `_uart_rx_idle_code`
UART Driver, [625](#)
- `_uart_timeout_isr_code`
UART Driver, [626](#)
- `_uart_tx1_isr_code`
UART Driver, [623](#)
- `_uart_tx2_isr_code`
UART Driver, [623](#)
- `_uart_wakeup_rx_complete_isr`
UART Driver, [622](#)
- `_uart_wakeup_rx_error_isr`
UART Driver, [622](#)
- `_uart_wakeup_rx_timeout_isr`
UART Driver, [622](#)
- `_uart_wakeup_tx1_isr`
UART Driver, [620](#)
- `_uart_wakeup_tx2_isr`
UART Driver, [620](#)

- `_usb_ep0in`
 - USB Driver, [674](#)
- `_usb_ep0out`
 - USB Driver, [675](#)
- `_usb_ep0setup`
 - USB Driver, [673](#)
- `_usb_isr_invoke_event_cb`
 - USB Driver, [656](#)
- `_usb_isr_invoke_in_cb`
 - USB Driver, [657](#)
- `_usb_isr_invoke_out_cb`
 - USB Driver, [657](#)
- `_usb_isr_invoke_setup_cb`
 - USB Driver, [657](#)
- `_usb_isr_invoke_sof_cb`
 - USB Driver, [656](#)
- `_usb_reset`
 - USB Driver, [671](#)
- `_usb_suspend`
 - USB Driver, [672](#)
- `_usb_wakeup`
 - USB Driver, [673](#)
- ADC Driver, [28](#)
 - `_adc_isr_error_code`, [33](#)
 - `_adc_isr_full_code`, [33](#)
 - `_adc_isr_half_code`, [32](#)
 - `_adc_reset_i`, [31](#)
 - `_adc_reset_s`, [31](#)
 - `_adc_timeout_isr`, [32](#)
 - `_adc_wakeup_isr`, [32](#)
 - ADC_ACTIVE, [35](#)
 - ADC_COMPLETE, [35](#)
 - ADC_ERR_AWD, [35](#)
 - ADC_ERR_DMAFAILURE, [35](#)
 - ADC_ERR_OVERFLOW, [35](#)
 - ADC_ERROR, [35](#)
 - ADC_READY, [35](#)
 - ADC_STOP, [35](#)
 - ADC_UNINIT, [35](#)
 - ADC_USE_MUTUAL_EXCLUSION, [31](#)
 - ADC_USE_WAIT, [31](#)
 - ADCD1, [44](#)
 - ADCDriver, [34](#)
 - `adc_channels_num_t`, [34](#)
 - `adc_llid_init`, [42](#)
 - `adc_llid_start`, [43](#)
 - `adc_llid_start_conversion`, [43](#)
 - `adc_llid_stop`, [43](#)
 - `adc_llid_stop_conversion`, [44](#)
 - `adcAcquireBus`, [41](#)
 - `adcConvert`, [40](#)
 - `adcInit`, [36](#)
 - `adcObjectInit`, [36](#)
 - `adcReleaseBus`, [42](#)
 - `adcStart`, [36](#)
 - `adcStartConversion`, [38](#)
 - `adcStartConversionI`, [39](#)
 - `adcStop`, [37](#)
 - `adcStopConversion`, [39](#)
 - `adcStopConversionI`, [40](#)
 - `adccallback_t`, [35](#)
 - `adcerror_t`, [35](#)
 - `adcerrorcallback_t`, [35](#)
 - `adcsample_t`, [34](#)
 - `adcstate_t`, [35](#)
 - PLATFORM_ADC_USE_ADC1, [34](#)
- ADC_ACTIVE
 - ADC Driver, [35](#)
- ADC_COMPLETE
 - ADC Driver, [35](#)
- ADC_ERR_AWD
 - ADC Driver, [35](#)
- ADC_ERR_DMAFAILURE
 - ADC Driver, [35](#)
- ADC_ERR_OVERFLOW
 - ADC Driver, [35](#)
- ADC_ERROR
 - ADC Driver, [35](#)
- ADC_READY
 - ADC Driver, [35](#)
- ADC_STOP
 - ADC Driver, [35](#)
- ADC_UNINIT
 - ADC Driver, [35](#)
- ADC_USE_MUTUAL_EXCLUSION
 - ADC Driver, [31](#)
 - Configuration, [330](#)
- ADC_USE_WAIT
 - ADC Driver, [31](#)
 - Configuration, [330](#)
- ADCCConfig, [691](#)
- ADCCConversionGroup, [692](#)
 - `circular`, [693](#)
 - `end_cb`, [693](#)
 - `error_cb`, [693](#)
 - `num_channels`, [693](#)
- ADCD1
 - ADC Driver, [44](#)
- ADCDriver, [693](#)
 - ADC Driver, [34](#)
 - `config`, [695](#)
 - `depth`, [695](#)
 - `grpp`, [695](#)
 - `mutex`, [695](#)
 - `samples`, [695](#)
 - `state`, [695](#)
 - `thread`, [695](#)
- ADVERTISE_100BASE4
 - MII/RMII Header, [389](#)
- ADVERTISE_100FULL
 - MII/RMII Header, [389](#)
- ADVERTISE_100HALF
 - MII/RMII Header, [389](#)
- ADVERTISE_10FULL
 - MII/RMII Header, [389](#)
- ADVERTISE_10HALF

- MII/RMII Header, [389](#)
- ADVERTISE_CSMA
 - MII/RMII Header, [389](#)
- ADVERTISE_LPACK
 - MII/RMII Header, [390](#)
- ADVERTISE_NPAGE
 - MII/RMII Header, [390](#)
- ADVERTISE_PAUSE_ASYM
 - MII/RMII Header, [390](#)
- ADVERTISE_PAUSE_CAP
 - MII/RMII Header, [390](#)
- ADVERTISE_RESV
 - MII/RMII Header, [390](#)
- ADVERTISE_RFAULT
 - MII/RMII Header, [390](#)
- ADVERTISE_SLCT
 - MII/RMII Header, [389](#)
- Abstract Files, [219](#)
 - _file_stream_data, [220](#)
 - _file_stream_methods, [220](#)
 - FILE_EOF, [220](#)
 - FILE_ERROR, [220](#)
 - FILE_OK, [220](#)
 - fileStreamClose, [222](#)
 - fileStreamGet, [222](#)
 - fileStreamGetError, [223](#)
 - fileStreamGetPosition, [223](#)
 - fileStreamGetSize, [223](#)
 - fileStreamPut, [221](#)
 - fileStreamRead, [221](#)
 - fileStreamSeek, [224](#)
 - fileStreamWrite, [220](#)
 - fileoffset_t, [224](#)
- Abstract I/O Block Device, [225](#)
 - _base_block_device_data, [226](#)
 - _base_block_device_methods, [226](#)
 - BLK_ACTIVE, [231](#)
 - BLK_CONNECTING, [231](#)
 - BLK_DISCONNECTING, [231](#)
 - BLK_READING, [231](#)
 - BLK_READY, [231](#)
 - BLK_STOP, [231](#)
 - BLK_SYNCING, [231](#)
 - BLK_UNINIT, [231](#)
 - BLK_WRITING, [231](#)
 - blkConnect, [228](#)
 - blkDisconnect, [229](#)
 - blkGetDriverState, [226](#)
 - blkGetInfo, [231](#)
 - blkIsInserted, [227](#)
 - blkIsTransferring, [227](#)
 - blkIsWriteProtected, [228](#)
 - blkRead, [229](#)
 - blkSync, [230](#)
 - blkWrite, [230](#)
 - blkstate_t, [231](#)
- Abstract I/O Channel, [210](#)
 - _base_asynchronous_channel_data, [216](#)
 - _base_asynchronous_channel_methods, [216](#)
 - _base_channel_data, [211](#)
 - _base_channel_methods, [211](#)
 - CHN_CONNECTED, [216](#)
 - CHN_DISCONNECTED, [216](#)
 - CHN_INPUT_AVAILABLE, [216](#)
 - CHN_NO_ERROR, [216](#)
 - CHN_OUTPUT_EMPTY, [216](#)
 - CHN_TRANSMISSION_END, [216](#)
 - chnAddFlagsI, [218](#)
 - chnControl, [215](#)
 - chnGetEventSource, [216](#)
 - chnGetTimeout, [212](#)
 - chnPutTimeout, [211](#)
 - chnRead, [213](#)
 - chnReadTimeout, [215](#)
 - chnWrite, [213](#)
 - chnWriteTimeout, [213](#)
- Abstract Peripheral Interfaces, [235](#)
- Abstract Streams, [256](#)
 - _base_sequential_stream_data, [257](#)
 - _base_sequential_stream_methods, [256](#)
 - streamGet, [258](#)
 - streamPut, [257](#)
 - streamRead, [257](#)
 - streamWrite, [257](#)
- accelerometerGetAxesNumber
 - Generic Accelerometer Interface, [24](#)
- accelerometerReadCooked
 - Generic Accelerometer Interface, [25](#)
- accelerometerReadRaw
 - Generic Accelerometer Interface, [24](#)
- accelerometerResetBias
 - Generic Accelerometer Interface, [26](#)
- accelerometerResetSensitivity
 - Generic Accelerometer Interface, [27](#)
- accelerometerSetBias
 - Generic Accelerometer Interface, [25](#)
- accelerometerSetSensitivity
 - Generic Accelerometer Interface, [26](#)
- adc_channels_num_t
 - ADC Driver, [34](#)
- adc_ild_init
 - ADC Driver, [42](#)
- adc_ild_start
 - ADC Driver, [43](#)
- adc_ild_start_conversion
 - ADC Driver, [43](#)
- adc_ild_stop
 - ADC Driver, [43](#)
- adc_ild_stop_conversion
 - ADC Driver, [44](#)
- adcAcquireBus
 - ADC Driver, [41](#)
- adcConvert
 - ADC Driver, [40](#)
- adclnit
 - ADC Driver, [36](#)

- adcObjectInit
 - ADC Driver, [36](#)
- adcReleaseBus
 - ADC Driver, [42](#)
- adcStart
 - ADC Driver, [36](#)
- adcStartConversion
 - ADC Driver, [38](#)
- adcStartConversionI
 - ADC Driver, [39](#)
- adcStop
 - ADC Driver, [37](#)
- adcStopConversion
 - ADC Driver, [39](#)
- adcStopConversionI
 - ADC Driver, [40](#)
- adccallback_t
 - ADC Driver, [35](#)
- adcerror_t
 - ADC Driver, [35](#)
- adcerrorcallback_t
 - ADC Driver, [35](#)
- adcsample_t
 - ADC Driver, [34](#)
- adcstate_t
 - ADC Driver, [35](#)
- address
 - flash_descriptor_t, [753](#)
 - USBDriver, [832](#)
- arg
 - palevent_t, [790](#)
- attributes
 - flash_descriptor_t, [752](#)
- BLK_ACTIVE
 - Abstract I/O Block Device, [231](#)
- BLK_CONNECTING
 - Abstract I/O Block Device, [231](#)
- BLK_DISCONNECTING
 - Abstract I/O Block Device, [231](#)
- BLK_READING
 - Abstract I/O Block Device, [231](#)
- BLK_READY
 - Abstract I/O Block Device, [231](#)
- BLK_STOP
 - Abstract I/O Block Device, [231](#)
- BLK_SYNCING
 - Abstract I/O Block Device, [231](#)
- BLK_UNINIT
 - Abstract I/O Block Device, [231](#)
- BLK_WRITING
 - Abstract I/O Block Device, [231](#)
- BMCR_ANENABLE
 - MII/RMII Header, [388](#)
- BMCR_ANRESTART
 - MII/RMII Header, [387](#)
- BMCR_CTST
 - MII/RMII Header, [387](#)
- BMCR_FULLDPLX
 - MII/RMII Header, [387](#)
- BMCR_ISOLATE
 - MII/RMII Header, [387](#)
- BMCR_LOOPBACK
 - MII/RMII Header, [388](#)
- BMCR_PDOWN
 - MII/RMII Header, [387](#)
- BMCR_RESET
 - MII/RMII Header, [388](#)
- BMCR_RESV
 - MII/RMII Header, [387](#)
- BMCR_SPEED100
 - MII/RMII Header, [388](#)
- BMSR_100BASE4
 - MII/RMII Header, [389](#)
- BMSR_100FULL
 - MII/RMII Header, [389](#)
- BMSR_100HALF
 - MII/RMII Header, [389](#)
- BMSR_10FULL
 - MII/RMII Header, [389](#)
- BMSR_10HALF
 - MII/RMII Header, [389](#)
- BMSR_ANEGCAPABLE
 - MII/RMII Header, [388](#)
- BMSR_ANEGCOMPLETE
 - MII/RMII Header, [388](#)
- BMSR_ERCAP
 - MII/RMII Header, [388](#)
- BMSR_JCD
 - MII/RMII Header, [388](#)
- BMSR_LSTATUS
 - MII/RMII Header, [388](#)
- BMSR_MFPRESUPPCAP
 - MII/RMII Header, [388](#)
- BMSR_RESV
 - MII/RMII Header, [388](#)
- BMSR_RFAULT
 - MII/RMII Header, [388](#)
- BQ_BUFFER_SIZE
 - I/O Buffers Queues, [189](#)
- bank0_sectors
 - MFSCConfig, [777](#)
- bank0_start
 - MFSCConfig, [777](#)
- bank1_sectors
 - MFSCConfig, [777](#)
- bank1_start
 - MFSCConfig, [777](#)
- bank_size
 - MFSCConfig, [777](#)
- barometerGetChannelsNumber
 - Generic Barometer Interface, [46](#)
- barometerReadCooked
 - Generic Barometer Interface, [47](#)
- barometerReadRaw
 - Generic Barometer Interface, [46](#)
- barometerResetBias

- Generic Barometer Interface, [48](#)
- barometerResetSensitivity
 - Generic Barometer Interface, [49](#)
- barometerSetBias
 - Generic Barometer Interface, [47](#)
- barometerSetSensitivity
 - Generic Barometer Interface, [48](#)
- Base Object, [232](#)
 - _base_object_data, [232](#)
 - _base_object_methods, [232](#)
 - objGetInstance, [232](#)
- BaseAccelerometer, [695](#)
 - vmt, [697](#)
- BaseAccelerometerVMT, [697](#)
- BaseAsynchronousChannel, [698](#)
 - vmt, [700](#)
- BaseAsynchronousChannelVMT, [700](#)
- BaseBarometer, [702](#)
 - vmt, [704](#)
- BaseBarometerVMT, [704](#)
- BaseBlockDevice, [705](#)
 - vmt, [707](#)
- BaseBlockDeviceVMT, [707](#)
- BaseChannel, [708](#)
 - vmt, [710](#)
- BaseChannelVMT, [710](#)
- BaseCompass, [712](#)
 - vmt, [713](#)
- BaseCompassVMT, [713](#)
- BaseFlash, [714](#)
 - vmt, [716](#)
- BaseFlashVMT, [716](#)
- BaseGyroscope, [717](#)
 - vmt, [718](#)
- BaseGyroscopeVMT, [718](#)
- BaseHygrometer, [719](#)
 - vmt, [720](#)
- BaseHygrometerVMT, [720](#)
- BaseObject, [721](#)
 - vmt, [722](#)
- BaseObjectVMT, [722](#)
- BaseSensor, [723](#)
 - vmt, [724](#)
- BaseSensorVMT, [724](#)
- BaseSequentialStream, [725](#)
 - vmt, [727](#)
- BaseSequentialStreamVMT, [727](#)
- BaseThermometer, [728](#)
 - vmt, [730](#)
- BaseThermometerVMT, [730](#)
- bcounter
 - io_buffers_queue, [764](#)
- bitsize_t
 - Crypto Driver, [78](#)
- blk_num
 - BlockDeviceInfo, [732](#)
- blk_size
 - BlockDeviceInfo, [732](#)
- blkConnect
 - Abstract I/O Block Device, [228](#)
- blkDisconnect
 - Abstract I/O Block Device, [229](#)
- blkGetDriverState
 - Abstract I/O Block Device, [226](#)
- blkGetInfo
 - Abstract I/O Block Device, [231](#)
- blkIsInserted
 - Abstract I/O Block Device, [227](#)
- blkIsTransferring
 - Abstract I/O Block Device, [227](#)
- blkIsWriteProtected
 - Abstract I/O Block Device, [228](#)
- blkRead
 - Abstract I/O Block Device, [229](#)
- blkSync
 - Abstract I/O Block Device, [230](#)
- blkWrite
 - Abstract I/O Block Device, [230](#)
- blkstate_t
 - Abstract I/O Block Device, [231](#)
- BlockDeviceInfo, [731](#)
 - blk_num, [732](#)
 - blk_size, [732](#)
- bn
 - io_buffers_queue, [765](#)
- bqGetLinkX
 - I/O Buffers Queues, [189](#)
- bqIsSuspendedX
 - I/O Buffers Queues, [190](#)
- bqResumeX
 - I/O Buffers Queues, [191](#)
- bqSizeX
 - I/O Buffers Queues, [189](#)
- bqSpacel
 - I/O Buffers Queues, [189](#)
- bqSuspendI
 - I/O Buffers Queues, [190](#)
- bqnotify_t
 - I/O Buffers Queues, [193](#)
- brdptr
 - io_buffers_queue, [764](#)
- bsize
 - io_buffers_queue, [765](#)
- btob
 - io_buffers_queue, [765](#)
- buffer
 - MFSDriver, [780](#)
- buffers
 - io_buffers_queue, [765](#)
- bulk_in
 - SerialUSBConfig, [811](#)
- bulk_out
 - SerialUSBConfig, [811](#)
- bus_width
 - SDConfig, [803](#)
- bwrptr

- io_buffers_queue, 764
- CAN Driver, 50
 - _can_error_isr, 54
 - _can_rx_full_isr, 54
 - _can_tx_empty_isr, 54
 - _can_wakeup_isr, 54
 - CAN_ANY_MAILBOX, 53
 - CAN_BUS_OFF_ERROR, 53
 - CAN_ENFORCE_USE_CALLBACKS, 53
 - CAN_FRAMING_ERROR, 53
 - CAN_LIMIT_ERROR, 53
 - CAN_LIMIT_WARNING, 53
 - CAN_MAILBOX_TO_MASK, 53
 - CAN_OVERFLOW_ERROR, 53
 - CAN_READY, 56
 - CAN_RX_MAILBOXES, 55
 - CAN_SLEEP, 56
 - CAN_STARTING, 56
 - CAN_STOP, 56
 - CAN_TX_MAILBOXES, 55
 - CAN_UNINIT, 56
 - CAN_USE_SLEEP_MODE, 53
 - CAND1, 67
 - CANDriver, 55
 - can_callback_t, 55
 - can_ild_init, 64
 - can_ild_is_rx_nonempty, 66
 - can_ild_is_tx_empty, 65
 - can_ild_receive, 66
 - can_ild_sleep, 66
 - can_ild_start, 64
 - can_ild_stop, 65
 - can_ild_transmit, 65
 - can_ild_wakeup, 67
 - canInit, 56
 - canObjectInit, 56
 - canReceive, 54
 - canReceiveTimeout, 61
 - canSleep, 62
 - canStart, 57
 - canStop, 57
 - canTransmit, 53
 - canTransmitTimeout, 60
 - canTryReceiveI, 59
 - canTryTransmitI, 58
 - canWakeup, 63
 - canmbx_t, 55
 - canstate_t, 55
 - PLATFORM_CAN_USE_CAN1, 55
- CAN_ANY_MAILBOX
 - CAN Driver, 53
- CAN_BUS_OFF_ERROR
 - CAN Driver, 53
- CAN_ENFORCE_USE_CALLBACKS
 - CAN Driver, 53
 - Configuration, 330
- CAN_FRAMING_ERROR
 - CAN Driver, 53
- CAN_LIMIT_ERROR
 - CAN Driver, 53
- CAN_LIMIT_WARNING
 - CAN Driver, 53
- CAN_MAILBOX_TO_MASK
 - CAN Driver, 53
- CAN_OVERFLOW_ERROR
 - CAN Driver, 53
- CAN_READY
 - CAN Driver, 56
- CAN_RX_MAILBOXES
 - CAN Driver, 55
- CAN_SLEEP
 - CAN Driver, 56
- CAN_STARTING
 - CAN Driver, 56
- CAN_STOP
 - CAN Driver, 56
- CAN_TX_MAILBOXES
 - CAN Driver, 55
- CAN_UNINIT
 - CAN Driver, 56
- CAN_USE_SLEEP_MODE
 - CAN Driver, 53
 - Configuration, 330
- CANConfig, 732
- CAND1
 - CAN Driver, 67
- CANDriver, 732
 - CAN Driver, 55
 - config, 734
 - error_event, 734
 - rxfull_event, 734
 - rxqueue, 734
 - sleep_event, 734
 - state, 734
 - txempty_event, 734
 - txqueue, 734
 - wakeup_event, 734
- CANRxFrame, 735
 - DLC, 736
 - data16, 736
 - data32, 736
 - data8, 736
 - EID, 736
 - FMI, 736
 - IDE, 736
 - RTR, 736
 - SID, 736
 - TIME, 736
- CANTxFrame, 737
 - DLC, 738
 - data16, 738
 - data32, 738
 - data8, 738
 - EID, 738
 - IDE, 738
 - RTR, 738

- SID, [738](#)
- CH_HAL_MAJOR
 - HAL Driver, [185](#)
- CH_HAL_MINOR
 - HAL Driver, [185](#)
- CH_HAL_PATCH
 - HAL Driver, [185](#)
- CH_HAL_STABLE
 - HAL Driver, [185](#)
- CHN_CONNECTED
 - Abstract I/O Channel, [216](#)
- CHN_DISCONNECTED
 - Abstract I/O Channel, [216](#)
- CHN_INPUT_AVAILABLE
 - Abstract I/O Channel, [216](#)
- CHN_NO_ERROR
 - Abstract I/O Channel, [216](#)
- CHN_OUTPUT_EMPTY
 - Abstract I/O Channel, [216](#)
- CHN_TRANSMISSION_END
 - Abstract I/O Channel, [216](#)
- CRY_ERR_INV_ALGO
 - Crypto Driver, [79](#)
- CRY_ERR_INV_KEY_ID
 - Crypto Driver, [79](#)
- CRY_ERR_INV_KEY_SIZE
 - Crypto Driver, [79](#)
- CRY_ERR_INV_KEY_TYPE
 - Crypto Driver, [79](#)
- CRY_NOERROR
 - Crypto Driver, [79](#)
- CRY_READY
 - Crypto Driver, [78](#)
- CRY_STOP
 - Crypto Driver, [78](#)
- CRY_UNINIT
 - Crypto Driver, [78](#)
- CRYConfig, [739](#)
- CRYD1
 - Crypto Driver, [126](#)
- CRYDriver, [739](#)
 - config, [740](#)
 - Crypto Driver, [78](#)
 - key0_buffer, [741](#)
 - key0_size, [741](#)
 - key0_type, [740](#)
 - state, [740](#)
- callback
 - GPTConfig, [755](#)
 - PWMChannelConfig, [791](#)
 - PWMConfig, [793](#)
- can_callback_t
 - CAN Driver, [55](#)
- can_ild_init
 - CAN Driver, [64](#)
- can_ild_is_rx_nonempty
 - CAN Driver, [66](#)
- can_ild_is_tx_empty
 - CAN Driver, [65](#)
- can_ild_receive
 - CAN Driver, [66](#)
- can_ild_sleep
 - CAN Driver, [66](#)
- can_ild_start
 - CAN Driver, [64](#)
- can_ild_stop
 - CAN Driver, [65](#)
- can_ild_transmit
 - CAN Driver, [65](#)
- can_ild_wakeup
 - CAN Driver, [67](#)
- canInit
 - CAN Driver, [56](#)
- canObjectInit
 - CAN Driver, [56](#)
- canReceive
 - CAN Driver, [54](#)
- canReceiveTimeout
 - CAN Driver, [61](#)
- canSleep
 - CAN Driver, [62](#)
- canStart
 - CAN Driver, [57](#)
- canStop
 - CAN Driver, [57](#)
- canTransmit
 - CAN Driver, [53](#)
- canTransmitTimeout
 - CAN Driver, [60](#)
- canTryReceiveI
 - CAN Driver, [59](#)
- canTryTransmitI
 - CAN Driver, [58](#)
- canWakeup
 - CAN Driver, [63](#)
- canmbx_t
 - CAN Driver, [55](#)
- canstate_t
 - CAN Driver, [55](#)
- cardmode
 - SDCDriver, [805](#)
- cb
 - EXTChannelConfig, [746](#)
 - event_source, [745](#)
 - palevent_t, [790](#)
- cdc_linecoding_t, [738](#)
- channels
 - EXTConfig, [747](#)
 - PWMConfig, [793](#)
 - PWMDriver, [795](#)
- chnAddFlagsI
 - Abstract I/O Channel, [218](#)
- chnControl
 - Abstract I/O Channel, [215](#)
- chnGetEventSource
 - Abstract I/O Channel, [216](#)

- chnGetTimeout
 - Abstract I/O Channel, [212](#)
- chnPutTimeout
 - Abstract I/O Channel, [211](#)
- chnRead
 - Abstract I/O Channel, [213](#)
- chnReadTimeout
 - Abstract I/O Channel, [215](#)
- chnWrite
 - Abstract I/O Channel, [213](#)
- chnWriteTimeout
 - Abstract I/O Channel, [213](#)
- circular
 - ADCConversionGroup, [693](#)
 - SPIConfig, [817](#)
- compassGetAxesNumber
 - Generic Compass Interface, [69](#)
- compassReadCooked
 - Generic Compass Interface, [70](#)
- compassReadRaw
 - Generic Compass Interface, [69](#)
- compassResetBias
 - Generic Compass Interface, [71](#)
- compassResetSensitivity
 - Generic Compass Interface, [72](#)
- compassSetBias
 - Generic Compass Interface, [70](#)
- compassSetSensitivity
 - Generic Compass Interface, [71](#)
- Complex Drivers, [337](#)
- config
 - ADCDriver, [695](#)
 - CANDriver, [734](#)
 - CRYDriver, [740](#)
 - DACDriver, [744](#)
 - EXTDriver, [748](#)
 - GPTDriver, [756](#)
 - I2CDriver, [758](#)
 - I2SDriver, [760](#)
 - ICUDriver, [763](#)
 - MACDriver, [771](#)
 - MFSDriver, [780](#)
 - MMCDriver, [783](#)
 - PWMDriver, [795](#)
 - QSPIDriver, [797](#)
 - SDCDriver, [804](#)
 - SPIDriver, [819](#)
 - UARTDriver, [823](#)
 - USBDriver, [831](#)
 - WDGDriver, [839](#)
- Configuration, [326](#)
 - ADC_USE_MUTUAL_EXCLUSION, [330](#)
 - ADC_USE_WAIT, [330](#)
 - CAN_ENFORCE_USE_CALLBACKS, [330](#)
 - CAN_USE_SLEEP_MODE, [330](#)
 - DAC_USE_MUTUAL_EXCLUSION, [331](#)
 - DAC_USE_WAIT, [331](#)
 - HAL_CRY_ENFORCE_FALLBACK, [331](#)
 - HAL_CRY_USE_FALLBACK, [330](#)
 - HAL_USE_ADC, [328](#)
 - HAL_USE_CAN, [328](#)
 - HAL_USE_CRY, [328](#)
 - HAL_USE_DAC, [328](#)
 - HAL_USE_EXT, [328](#)
 - HAL_USE_GPT, [328](#)
 - HAL_USE_I2C, [328](#)
 - HAL_USE_I2S, [329](#)
 - HAL_USE_ICU, [329](#)
 - HAL_USE_MAC, [329](#)
 - HAL_USE_MMC_SPI, [329](#)
 - HAL_USE_PAL, [328](#)
 - HAL_USE_PWM, [329](#)
 - HAL_USE_QSPI, [329](#)
 - HAL_USE_RTC, [329](#)
 - HAL_USE_SDC, [329](#)
 - HAL_USE_SERIAL_USB, [329](#)
 - HAL_USE_SERIAL, [329](#)
 - HAL_USE_SPI, [329](#)
 - HAL_USE_UART, [329](#)
 - HAL_USE_USB, [330](#)
 - HAL_USE_WDG, [330](#)
 - I2C_USE_MUTUAL_EXCLUSION, [331](#)
 - MAC_USE_EVENTS, [331](#)
 - MAC_USE_ZERO_COPY, [331](#)
 - MMC_NICE_WAITING, [331](#)
 - PAL_USE_CALLBACKS, [330](#)
 - PAL_USE_WAIT, [330](#)
 - QSPI_USE_MUTUAL_EXCLUSION, [332](#)
 - QSPI_USE_WAIT, [331](#)
 - SDC_INIT_OCR_V20, [332](#)
 - SDC_INIT_OCR, [332](#)
 - SDC_INIT_RETRY, [332](#)
 - SDC_MMC_SUPPORT, [332](#)
 - SDC_NICE_WAITING, [332](#)
 - SERIAL_BUFFERS_SIZE, [332](#)
 - SERIAL_DEFAULT_BITRATE, [332](#)
 - SERIAL_USB_BUFFERS_NUMBER, [333](#)
 - SERIAL_USB_BUFFERS_SIZE, [333](#)
 - SPI_SELECT_MODE, [333](#)
 - SPI_USE_CIRCULAR, [333](#)
 - SPI_USE_MUTUAL_EXCLUSION, [333](#)
 - SPI_USE_WAIT, [333](#)
 - UART_USE_MUTUAL_EXCLUSION, [334](#)
 - UART_USE_WAIT, [334](#)
 - USB_USE_WAIT, [334](#)
- configuration
 - USBDriver, [832](#)
- counter
 - mfs_bank_header_t, [774](#)
- crc
 - mfs_bank_header_t, [774](#)
 - mfs_data_header_t, [775](#)
- crc7
 - MMC over SPI Driver, [396](#)
- crc7_lookup_table
 - MMC over SPI Driver, [412](#)

- cry_algo_aes
 - Crypto Driver, [79](#)
- cry_algo_des
 - Crypto Driver, [79](#)
- cry_lld_SHA1_final
 - Crypto Driver, [122](#)
- cry_lld_SHA1_init
 - Crypto Driver, [121](#)
- cry_lld_SHA1_update
 - Crypto Driver, [121](#)
- cry_lld_SHA256_final
 - Crypto Driver, [123](#)
- cry_lld_SHA256_init
 - Crypto Driver, [122](#)
- cry_lld_SHA256_update
 - Crypto Driver, [123](#)
- cry_lld_SHA512_final
 - Crypto Driver, [125](#)
- cry_lld_SHA512_init
 - Crypto Driver, [124](#)
- cry_lld_SHA512_update
 - Crypto Driver, [124](#)
- cry_lld_TRNG
 - Crypto Driver, [125](#)
- cry_lld_decrypt_AES_CBC
 - Crypto Driver, [112](#)
- cry_lld_decrypt_AES_CFB
 - Crypto Driver, [113](#)
- cry_lld_decrypt_AES_CTR
 - Crypto Driver, [114](#)
- cry_lld_decrypt_AES_ECB
 - Crypto Driver, [110](#)
- cry_lld_decrypt_AES_GCM
 - Crypto Driver, [116](#)
- cry_lld_decrypt_AES
 - Crypto Driver, [109](#)
- cry_lld_decrypt_DES_CBC
 - Crypto Driver, [120](#)
- cry_lld_decrypt_DES_ECB
 - Crypto Driver, [119](#)
- cry_lld_decrypt_DES
 - Crypto Driver, [117](#)
- cry_lld_encrypt_AES_CBC
 - Crypto Driver, [111](#)
- cry_lld_encrypt_AES_CFB
 - Crypto Driver, [112](#)
- cry_lld_encrypt_AES_CTR
 - Crypto Driver, [114](#)
- cry_lld_encrypt_AES_ECB
 - Crypto Driver, [110](#)
- cry_lld_encrypt_AES_GCM
 - Crypto Driver, [115](#)
- cry_lld_encrypt_AES
 - Crypto Driver, [108](#)
- cry_lld_encrypt_DES_CBC
 - Crypto Driver, [119](#)
- cry_lld_encrypt_DES_ECB
 - Crypto Driver, [118](#)
- cry_lld_encrypt_DES
 - Crypto Driver, [117](#)
- cry_lld_init
 - Crypto Driver, [107](#)
- cry_lld_loadkey
 - Crypto Driver, [108](#)
- cry_lld_start
 - Crypto Driver, [107](#)
- cry_lld_stop
 - Crypto Driver, [108](#)
- cryDecryptAES_CBC
 - Crypto Driver, [87](#)
- cryDecryptAES_CFB
 - Crypto Driver, [89](#)
- cryDecryptAES_CTR
 - Crypto Driver, [91](#)
- cryDecryptAES_ECB
 - Crypto Driver, [85](#)
- cryDecryptAES_GCM
 - Crypto Driver, [93](#)
- cryDecryptAES
 - Crypto Driver, [83](#)
- cryDecryptDES_CBC
 - Crypto Driver, [99](#)
- cryDecryptDES_ECB
 - Crypto Driver, [97](#)
- cryDecryptDES
 - Crypto Driver, [95](#)
- cryEncryptAES_CBC
 - Crypto Driver, [86](#)
- cryEncryptAES_CFB
 - Crypto Driver, [88](#)
- cryEncryptAES_CTR
 - Crypto Driver, [90](#)
- cryEncryptAES_ECB
 - Crypto Driver, [84](#)
- cryEncryptAES_GCM
 - Crypto Driver, [92](#)
- cryEncryptAES
 - Crypto Driver, [82](#)
- cryEncryptDES_CBC
 - Crypto Driver, [98](#)
- cryEncryptDES_ECB
 - Crypto Driver, [96](#)
- cryEncryptDES
 - Crypto Driver, [94](#)
- cryInit
 - Crypto Driver, [79](#)
- cryLoadTransientKey
 - Crypto Driver, [81](#)
- cryObjectInit
 - Crypto Driver, [79](#)
- crySHA1Final
 - Crypto Driver, [101](#)
- crySHA1Init
 - Crypto Driver, [100](#)
- crySHA1Update
 - Crypto Driver, [100](#)

- crySHA256Final
 - Crypto Driver, [103](#)
- crySHA256Init
 - Crypto Driver, [102](#)
- crySHA256Update
 - Crypto Driver, [103](#)
- crySHA512Final
 - Crypto Driver, [106](#)
- crySHA512Init
 - Crypto Driver, [104](#)
- crySHA512Update
 - Crypto Driver, [105](#)
- cryStart
 - Crypto Driver, [80](#)
- cryStop
 - Crypto Driver, [80](#)
- cryTRNG
 - Crypto Driver, [107](#)
- cryalgorithm_t
 - Crypto Driver, [79](#)
- cryerror_t
 - Crypto Driver, [78](#)
- crykey_t
 - Crypto Driver, [78](#)
- Crypto Driver, [73](#)
 - bitsize_t, [78](#)
 - CRY_ERR_INV_ALGO, [79](#)
 - CRY_ERR_INV_KEY_ID, [79](#)
 - CRY_ERR_INV_KEY_SIZE, [79](#)
 - CRY_ERR_INV_KEY_TYPE, [79](#)
 - CRY_NOERROR, [79](#)
 - CRY_READY, [78](#)
 - CRY_STOP, [78](#)
 - CRY_UNINIT, [78](#)
 - CRYD1, [126](#)
 - CRYDriver, [78](#)
 - cry_algo_aes, [79](#)
 - cry_algo_des, [79](#)
 - cry_llc_SHA1_final, [122](#)
 - cry_llc_SHA1_init, [121](#)
 - cry_llc_SHA1_update, [121](#)
 - cry_llc_SHA256_final, [123](#)
 - cry_llc_SHA256_init, [122](#)
 - cry_llc_SHA256_update, [123](#)
 - cry_llc_SHA512_final, [125](#)
 - cry_llc_SHA512_init, [124](#)
 - cry_llc_SHA512_update, [124](#)
 - cry_llc_TRNG, [125](#)
 - cry_llc_decrypt_AES_CBC, [112](#)
 - cry_llc_decrypt_AES_CFB, [113](#)
 - cry_llc_decrypt_AES_CTR, [114](#)
 - cry_llc_decrypt_AES_ECB, [110](#)
 - cry_llc_decrypt_AES_GCM, [116](#)
 - cry_llc_decrypt_AES, [109](#)
 - cry_llc_decrypt_DES_CBC, [120](#)
 - cry_llc_decrypt_DES_ECB, [119](#)
 - cry_llc_decrypt_DES, [117](#)
 - cry_llc_encrypt_AES_CBC, [111](#)
 - cry_llc_encrypt_AES_CFB, [112](#)
 - cry_llc_encrypt_AES_CTR, [114](#)
 - cry_llc_encrypt_AES_ECB, [110](#)
 - cry_llc_encrypt_AES_GCM, [115](#)
 - cry_llc_encrypt_AES, [108](#)
 - cry_llc_encrypt_DES_CBC, [119](#)
 - cry_llc_encrypt_DES_ECB, [118](#)
 - cry_llc_encrypt_DES, [117](#)
 - cry_llc_init, [107](#)
 - cry_llc_loadkey, [108](#)
 - cry_llc_start, [107](#)
 - cry_llc_stop, [108](#)
 - cryDecryptAES_CBC, [87](#)
 - cryDecryptAES_CFB, [89](#)
 - cryDecryptAES_CTR, [91](#)
 - cryDecryptAES_ECB, [85](#)
 - cryDecryptAES_GCM, [93](#)
 - cryDecryptAES, [83](#)
 - cryDecryptDES_CBC, [99](#)
 - cryDecryptDES_ECB, [97](#)
 - cryDecryptDES, [95](#)
 - cryEncryptAES_CBC, [86](#)
 - cryEncryptAES_CFB, [88](#)
 - cryEncryptAES_CTR, [90](#)
 - cryEncryptAES_ECB, [84](#)
 - cryEncryptAES_GCM, [92](#)
 - cryEncryptAES, [82](#)
 - cryEncryptDES_CBC, [98](#)
 - cryEncryptDES_ECB, [96](#)
 - cryEncryptDES, [94](#)
 - cryInit, [79](#)
 - cryLoadTransientKey, [81](#)
 - cryObjectInit, [79](#)
 - crySHA1Final, [101](#)
 - crySHA1Init, [100](#)
 - crySHA1Update, [100](#)
 - crySHA256Final, [103](#)
 - crySHA256Init, [102](#)
 - crySHA256Update, [103](#)
 - crySHA512Final, [106](#)
 - crySHA512Init, [104](#)
 - crySHA512Update, [105](#)
 - cryStart, [80](#)
 - cryStop, [80](#)
 - cryTRNG, [107](#)
 - cryalgorithm_t, [79](#)
 - cryerror_t, [78](#)
 - crykey_t, [78](#)
 - crystate_t, [78](#)
 - HAL_CRY_ENFORCE_FALLBACK, [77](#)
 - HAL_CRY_MAX_KEY_SIZE, [77](#)
 - HAL_CRY_USE_FALLBACK, [77](#)
 - PLATFORM_CRY_USE_CRY1, [78](#)
 - crystate_t
 - Crypto Driver, [78](#)
 - current_bank
 - MFSDriver, [780](#)
 - current_counter

- MFSDriver, 780
- DAC Driver, 127
 - _dac_isr_error_code, 132
 - _dac_isr_full_code, 131
 - _dac_isr_half_code, 131
 - _dac_reset_i, 130
 - _dac_reset_s, 130
 - _dac_timeout_isr, 131
 - _dac_wait_s, 129
 - _dac_wakeup_isr, 130
 - DAC_ACTIVE, 134
 - DAC_COMPLETE, 134
 - DAC_ERR_DMAFAILURE, 134
 - DAC_ERR_UNDERFLOW, 134
 - DAC_ERROR, 134
 - DAC_MAX_CHANNELS, 133
 - DAC_READY, 134
 - DAC_STOP, 134
 - DAC_UNINIT, 134
 - DAC_USE_MUTUAL_EXCLUSION, 129
 - DAC_USE_WAIT, 129
 - DACD1, 144
 - DACDriver, 133
 - dac_llid_init, 142
 - dac_llid_put_channel, 143
 - dac_llid_start, 142
 - dac_llid_start_conversion, 143
 - dac_llid_stop, 143
 - dac_llid_stop_conversion, 143
 - dacAcquireBus, 141
 - dacConvert, 140
 - daclnit, 134
 - dacObjectInit, 135
 - dacPutChannelX, 136
 - dacReleaseBus, 141
 - dacStart, 135
 - dacStartConversion, 137
 - dacStartConversionI, 138
 - dacStop, 136
 - dacStopConversion, 138
 - dacStopConversionI, 139
 - daccallback_t, 133
 - dacchannel_t, 133
 - dacerror_t, 134
 - dacerrorcallback_t, 133
 - dacsample_t, 133
 - dacstate_t, 134
 - PLATFORM_DAC_USE_DAC1, 133
- DAC_ACTIVE
 - DAC Driver, 134
- DAC_COMPLETE
 - DAC Driver, 134
- DAC_ERR_DMAFAILURE
 - DAC Driver, 134
- DAC_ERR_UNDERFLOW
 - DAC Driver, 134
- DAC_ERROR
 - DAC Driver, 134
- DAC_MAX_CHANNELS
 - DAC Driver, 133
- DAC_READY
 - DAC Driver, 134
- DAC_STOP
 - DAC Driver, 134
- DAC_UNINIT
 - DAC Driver, 134
- DAC_USE_MUTUAL_EXCLUSION
 - Configuration, 331
 - DAC Driver, 129
- DAC_USE_WAIT
 - Configuration, 331
 - DAC Driver, 129
- DACConfig, 741
- DACConversionGroup, 741
 - end_cb, 742
 - error_cb, 742
 - num_channels, 742
- DACD1
 - DAC Driver, 144
- DACDriver, 743
 - config, 744
 - DAC Driver, 133
 - depth, 744
 - grpp, 744
 - mutex, 744
 - samples, 744
 - state, 744
 - thread, 744
- DLC
 - CANRxFrame, 736
 - CANTxFrame, 738
- dac_llid_init
 - DAC Driver, 142
- dac_llid_put_channel
 - DAC Driver, 143
- dac_llid_start
 - DAC Driver, 142
- dac_llid_start_conversion
 - DAC Driver, 143
- dac_llid_stop
 - DAC Driver, 143
- dac_llid_stop_conversion
 - DAC Driver, 143
- dacAcquireBus
 - DAC Driver, 141
- dacConvert
 - DAC Driver, 140
- daclnit
 - DAC Driver, 134
- dacObjectInit
 - DAC Driver, 135
- dacPutChannelX
 - DAC Driver, 136
- dacReleaseBus
 - DAC Driver, 141
- dacStart

- DAC Driver, [135](#)
- dacStartConversion
 - DAC Driver, [137](#)
- dacStartConversionI
 - DAC Driver, [138](#)
- dacStop
 - DAC Driver, [136](#)
- dacStopConversion
 - DAC Driver, [138](#)
- dacStopConversionI
 - DAC Driver, [139](#)
- daccallback_t
 - DAC Driver, [133](#)
- dacchannel_t
 - DAC Driver, [133](#)
- dacerror_t
 - DAC Driver, [134](#)
- dacerrorcallback_t
 - DAC Driver, [133](#)
- dacsample_t
 - DAC Driver, [133](#)
- dacstate_t
 - DAC Driver, [134](#)
- data16
 - CANRxFrame, [736](#)
 - CANTxFrame, [738](#)
- data32
 - CANRxFrame, [736](#)
 - CANTxFrame, [738](#)
- data8
 - CANRxFrame, [736](#)
 - CANTxFrame, [738](#)
- day
 - RTCDateTime, [799](#)
- dayofweek
 - RTCDateTime, [799](#)
- default_config
 - Serial Driver, [564](#)
- default_handler
 - USB Driver, [662](#)
- depth
 - ADCDriver, [695](#)
 - DACDriver, [744](#)
- descriptors
 - MFSDriver, [780](#)
- detect_bus_clk
 - SDC Driver, [529](#)
- dstflag
 - RTCDateTime, [799](#)
- EID
 - CANRxFrame, [736](#)
 - CANTxFrame, [738](#)
- EP_STATUS_ACTIVE
 - USB Driver, [661](#)
- EP_STATUS_DISABLED
 - USB Driver, [661](#)
- EP_STATUS_STALLED
 - USB Driver, [661](#)
- ETHD1
 - MAC Driver, [322](#)
- EXPANSION_ENABLEPAGE
 - MII/RMII Header, [391](#)
- EXPANSION_LCWP
 - MII/RMII Header, [391](#)
- EXPANSION_MFAULTS
 - MII/RMII Header, [391](#)
- EXPANSION_NPCAPABLE
 - MII/RMII Header, [391](#)
- EXPANSION_NWAY
 - MII/RMII Header, [391](#)
- EXPANSION_RESV
 - MII/RMII Header, [391](#)
- EXT Driver, [145](#)
 - EXT_ACTIVE, [150](#)
 - EXT_CH_MODE_AUTOSTART, [148](#)
 - EXT_CH_MODE_BOTH_EDGES, [147](#)
 - EXT_CH_MODE_DISABLED, [147](#)
 - EXT_CH_MODE_EDGES_MASK, [147](#)
 - EXT_CH_MODE_FALLING_EDGE, [147](#)
 - EXT_CH_MODE_LOW_LEVEL, [148](#)
 - EXT_CH_MODE_RISING_EDGE, [147](#)
 - EXT_MAX_CHANNELS, [149](#)
 - EXT_STOP, [150](#)
 - EXT_UNINIT, [150](#)
 - EXTD1, [155](#)
 - EXTDriver, [149](#)
 - expchannel_t, [149](#)
 - ext_llc_channel_disable, [155](#)
 - ext_llc_channel_enable, [155](#)
 - ext_llc_init, [154](#)
 - ext_llc_start, [154](#)
 - ext_llc_stop, [154](#)
 - extChannelDisable, [152](#)
 - extChannelDisableI, [148](#)
 - extChannelEnable, [152](#)
 - extChannelEnableI, [148](#)
 - extInit, [150](#)
 - extObjectInit, [150](#)
 - extSetChannelMode, [148](#)
 - extSetChannelModel, [153](#)
 - extStart, [150](#)
 - extStop, [151](#)
 - extcallback_t, [149](#)
 - extstate_t, [149](#)
 - PLATFORM_EXT_USE_EXT1, [149](#)
- EXT_ACTIVE
 - EXT Driver, [150](#)
- EXT_CH_MODE_AUTOSTART
 - EXT Driver, [148](#)
- EXT_CH_MODE_BOTH_EDGES
 - EXT Driver, [147](#)
- EXT_CH_MODE_DISABLED
 - EXT Driver, [147](#)
- EXT_CH_MODE_EDGES_MASK
 - EXT Driver, [147](#)
- EXT_CH_MODE_FALLING_EDGE

- EXT Driver, 147
- EXT_CH_MODE_LOW_LEVEL
 - EXT Driver, 148
- EXT_CH_MODE_RISING_EDGE
 - EXT Driver, 147
- EXT_MAX_CHANNELS
 - EXT Driver, 149
- EXT_STOP
 - EXT Driver, 150
- EXT_UNINIT
 - EXT Driver, 150
- EXTChannelConfig, 745
 - cb, 746
 - mode, 746
- EXTConfig, 747
 - channels, 747
- EXTD1
 - EXT Driver, 155
- EXTDriver, 748
 - config, 748
 - EXT Driver, 149
 - state, 748
- early
 - UARTDriver, 823
- enabled
 - PWMDriver, 795
- end_cb
 - ADCCConversionGroup, 693
 - DACCConversionGroup, 742
 - I2SConfig, 759
 - QSPICongig, 796
 - SPICongig, 817
- ep0_state
 - USB Driver, 681
- ep0config
 - USB Driver, 682
- ep0endcb
 - USBDriver, 832
- ep0n
 - USBDriver, 832
- ep0next
 - USBDriver, 832
- ep0state
 - USBDriver, 832
- ep_mode
 - USBEndpointConfig, 835
- epc
 - USBDriver, 831
- erased
 - MFSCongig, 777
- error_cb
 - ADCCConversionGroup, 693
 - DACCConversionGroup, 742
- error_event
 - CANDriver, 734
- errors
 - I2CDriver, 758
 - SDCDriver, 805
- event_cb
 - USBConfig, 829
- event_source, 744
 - cb, 745
 - flags, 745
 - param, 745
- event_source_t
 - OSAL, 353
- eventcallback_t
 - OSAL, 353
- eventflags_t
 - OSAL, 353
- expchannel_t
 - EXT Driver, 149
- ext_lld_channel_disable
 - EXT Driver, 155
- ext_lld_channel_enable
 - EXT Driver, 155
- ext_lld_init
 - EXT Driver, 154
- ext_lld_start
 - EXT Driver, 154
- ext_lld_stop
 - EXT Driver, 154
- extChannelDisable
 - EXT Driver, 152
- extChannelDisableI
 - EXT Driver, 148
- extChannelEnable
 - EXT Driver, 152
- extChannelEnableI
 - EXT Driver, 148
- extInit
 - EXT Driver, 150
- extObjectInit
 - EXT Driver, 150
- extSetChannelMode
 - EXT Driver, 148
- extSetChannelModel
 - EXT Driver, 153
- extStart
 - EXT Driver, 150
- extStop
 - EXT Driver, 151
- extcallback_t
 - EXT Driver, 149
- extstate_t
 - EXT Driver, 149
- FILE_EOF
 - Abstract Files, 220
- FILE_ERROR
 - Abstract Files, 220
- FILE_OK
 - Abstract Files, 220
- FMI
 - CANRxFrame, 736
- FileStream, 749
 - vmt, 750

- fileStreamClose
 - Abstract Files, [222](#)
- fileStreamGet
 - Abstract Files, [222](#)
- fileStreamGetError
 - Abstract Files, [223](#)
- fileStreamGetPosition
 - Abstract Files, [223](#)
- fileStreamGetSize
 - Abstract Files, [223](#)
- fileStreamPut
 - Abstract Files, [221](#)
- fileStreamRead
 - Abstract Files, [221](#)
- fileStreamSeek
 - Abstract Files, [224](#)
- FileStreamVMT, [750](#)
- fileStreamWrite
 - Abstract Files, [220](#)
- fileoffset_t
 - Abstract Files, [224](#)
- flags
 - event_source, [745](#)
- flash_descriptor_t, [751](#)
 - address, [753](#)
 - attributes, [752](#)
 - page_size, [752](#)
 - sectors, [753](#)
 - sectors_count, [753](#)
 - sectors_size, [753](#)
- flash_error_t
 - Generic Flash Driver Interface, [162](#)
- flash_offset_t
 - Generic Flash Driver Interface, [162](#)
- flash_sector_descriptor_t, [753](#)
 - offset, [754](#)
 - size, [754](#)
- flash_sector_t
 - Generic Flash Driver Interface, [162](#)
- flash_state_t
 - Generic Flash Driver Interface, [162](#)
- flashGetDescriptor
 - Generic Flash Driver Interface, [158](#)
- flashGetSectorOffset
 - Generic Flash Driver Interface, [163](#)
- flashGetSectorSize
 - Generic Flash Driver Interface, [163](#)
- flashProgram
 - Generic Flash Driver Interface, [159](#)
- flashQueryErase
 - Generic Flash Driver Interface, [161](#)
- flashRead
 - Generic Flash Driver Interface, [159](#)
- flashStartEraseAll
 - Generic Flash Driver Interface, [160](#)
- flashStartEraseSector
 - Generic Flash Driver Interface, [160](#)
- flashVerifyErase
 - Generic Flash Driver Interface, [161](#)
- flashWaitErase
 - Generic Flash Driver Interface, [162](#)
- flashp
 - MFSCConfig, [777](#)
- frequency
 - GPTConfig, [755](#)
 - ICUConfig, [761](#)
 - PWMConfig, [793](#)
- GPT Driver, [164](#)
 - GPT_CONTINUOUS, [169](#)
 - GPT_ONESHOT, [169](#)
 - GPT_READY, [169](#)
 - GPT_STOP, [169](#)
 - GPT_UNINIT, [169](#)
 - GPTD1, [178](#)
 - GPTDriver, [169](#)
 - gpt_lld_change_interval, [168](#)
 - gpt_lld_init, [176](#)
 - gpt_lld_polled_delay, [177](#)
 - gpt_lld_start, [176](#)
 - gpt_lld_start_timer, [177](#)
 - gpt_lld_stop, [176](#)
 - gpt_lld_stop_timer, [177](#)
 - gptChangeInterval, [171](#)
 - gptChangeIntervall, [167](#)
 - gptGetCounterX, [167](#)
 - gptGetIntervalX, [167](#)
 - gptInit, [169](#)
 - gptObjectInit, [170](#)
 - gptPolledDelay, [175](#)
 - gptStart, [170](#)
 - gptStartContinuous, [172](#)
 - gptStartContinuousI, [173](#)
 - gptStartOneShot, [173](#)
 - gptStartOneShotI, [174](#)
 - gptStop, [171](#)
 - gptStopTimer, [174](#)
 - gptStopTimerI, [175](#)
 - gptcallback_t, [169](#)
 - gptcnt_t, [169](#)
 - gptfreq_t, [169](#)
 - gptstate_t, [169](#)
 - PLATFORM_GPT_USE_GPT1, [168](#)
- GPT_CONTINUOUS
 - GPT Driver, [169](#)
- GPT_ONESHOT
 - GPT Driver, [169](#)
- GPT_READY
 - GPT Driver, [169](#)
- GPT_STOP
 - GPT Driver, [169](#)
- GPT_UNINIT
 - GPT Driver, [169](#)
- GPTConfig, [754](#)
 - callback, [755](#)
 - frequency, [755](#)
- GPTD1

- GPT Driver, 178
- GPTDriver, 755
 - config, 756
 - GPT Driver, 169
 - state, 756
- Generic Accelerometer Interface, 23
 - _base_accelerometer_data, 24
 - _base_accelerometer_methods, 24
 - _base_accelerometer_methods_alone, 24
 - accelerometerGetAxesNumber, 24
 - accelerometerReadCooked, 25
 - accelerometerReadRaw, 24
 - accelerometerResetBias, 26
 - accelerometerResetSensitivity, 27
 - accelerometerSetBias, 25
 - accelerometerSetSensitivity, 26
- Generic Barometer Interface, 45
 - _base_barometer_data, 46
 - _base_barometer_methods, 46
 - _base_barometer_methods_alone, 45
 - barometerGetChannelsNumber, 46
 - barometerReadCooked, 47
 - barometerReadRaw, 46
 - barometerResetBias, 48
 - barometerResetSensitivity, 49
 - barometerSetBias, 47
 - barometerSetSensitivity, 48
- Generic Compass Interface, 68
 - _base_compass_data, 69
 - _base_compass_methods, 69
 - _base_compass_methods_alone, 68
 - compassGetAxesNumber, 69
 - compassReadCooked, 70
 - compassReadRaw, 69
 - compassResetBias, 71
 - compassResetSensitivity, 72
 - compassSetBias, 70
 - compassSetSensitivity, 71
- Generic Flash Driver Interface, 156
 - _base_flash_data, 158
 - _base_flash_methods, 158
 - _base_flash_methods_alone, 158
 - flash_error_t, 162
 - flash_offset_t, 162
 - flash_sector_t, 162
 - flash_state_t, 162
 - flashGetDescriptor, 158
 - flashGetSectorOffset, 163
 - flashGetSectorSize, 163
 - flashProgram, 159
 - flashQueryErase, 161
 - flashRead, 159
 - flashStartEraseAll, 160
 - flashStartEraseSector, 160
 - flashVerifyErase, 161
 - flashWaitErase, 162
 - getBaseFlash, 158
- Generic Gyroscope Interface, 179
 - _base_gyroscope_data, 180
 - _base_gyroscope_methods, 180
 - _base_gyroscope_methods_alone, 179
 - gyroscopeGetAxesNumber, 180
 - gyroscopeReadCooked, 181
 - gyroscopeReadRaw, 180
 - gyroscopeResetBias, 182
 - gyroscopeResetSensitivity, 183
 - gyroscopeSampleBias, 181
 - gyroscopeSetBias, 182
 - gyroscopeSetSensitivity, 183
- Generic Hygrometer Interface, 259
 - _base_hygrometer_data, 260
 - _base_hygrometer_methods, 260
 - _base_hygrometer_methods_alone, 259
 - hygrometerGetChannelsNumber, 260
 - hygrometerReadCooked, 261
 - hygrometerReadRaw, 260
 - hygrometerResetBias, 262
 - hygrometerResetSensitivity, 263
 - hygrometerSetBias, 261
 - hygrometerSetSensitivity, 262
- Generic Sensor Interface, 546
 - _base_sensor_data, 547
 - _base_sensor_methods, 546
 - _base_sensor_methods_alone, 546
 - sensorGetChannelNumber, 547
 - sensorReadCooked, 548
 - sensorReadRaw, 547
- Generic Thermometer Interface, 610
 - _base_thermometer_data, 611
 - _base_thermometer_methods, 611
 - _base_thermometer_methods_alone, 610
 - thermometerGetChannelsNumber, 611
 - thermometerReadCooked, 612
 - thermometerReadRaw, 611
 - thermometerResetBias, 613
 - thermometerResetSensitivity, 614
 - thermometerSetBias, 612
 - thermometerSetSensitivity, 613
- get_descriptor_cb
 - USBConfig, 829
- getBaseFlash
 - Generic Flash Driver Interface, 158
- gpt_ild_change_interval
 - GPT Driver, 168
- gpt_ild_init
 - GPT Driver, 176
- gpt_ild_polled_delay
 - GPT Driver, 177
- gpt_ild_start
 - GPT Driver, 176
- gpt_ild_start_timer
 - GPT Driver, 177
- gpt_ild_stop
 - GPT Driver, 176
- gpt_ild_stop_timer
 - GPT Driver, 177

- gptChangeInterval
 - GPT Driver, [171](#)
- gptChangeIntervall
 - GPT Driver, [167](#)
- gptGetCounterX
 - GPT Driver, [167](#)
- gptGetIntervalX
 - GPT Driver, [167](#)
- gptInit
 - GPT Driver, [169](#)
- gptObjectInit
 - GPT Driver, [170](#)
- gptPolledDelay
 - GPT Driver, [175](#)
- gptStart
 - GPT Driver, [170](#)
- gptStartContinuous
 - GPT Driver, [172](#)
- gptStartContinuousl
 - GPT Driver, [173](#)
- gptStartOneShot
 - GPT Driver, [173](#)
- gptStartOneShotl
 - GPT Driver, [174](#)
- gptStop
 - GPT Driver, [171](#)
- gptStopTimer
 - GPT Driver, [174](#)
- gptStopTimerl
 - GPT Driver, [175](#)
- gptcallback_t
 - GPT Driver, [169](#)
- gptcnt_t
 - GPT Driver, [169](#)
- gptfreq_t
 - GPT Driver, [169](#)
- gptstate_t
 - GPT Driver, [169](#)
- grpp
 - ADCDriver, [695](#)
 - DACDriver, [744](#)
- gyroscopeGetAxesNumber
 - Generic Gyroscope Interface, [180](#)
- gyroscopeReadCooked
 - Generic Gyroscope Interface, [181](#)
- gyroscopeReadRaw
 - Generic Gyroscope Interface, [180](#)
- gyroscopeResetBias
 - Generic Gyroscope Interface, [182](#)
- gyroscopeResetSensitivity
 - Generic Gyroscope Interface, [183](#)
- gyroscopeSampleBias
 - Generic Gyroscope Interface, [181](#)
- gyroscopeSetBias
 - Generic Gyroscope Interface, [182](#)
- gyroscopeSetSensitivity
 - Generic Gyroscope Interface, [183](#)
- HAL Driver, [184](#)
- _CHIBIOS_HAL_, [185](#)
 - CH_HAL_MAJOR, [185](#)
 - CH_HAL_MINOR, [185](#)
 - CH_HAL_PATCH, [185](#)
 - CH_HAL_STABLE, [185](#)
 - HAL_VERSION, [185](#)
 - hal_ild_init, [186](#)
 - halInit, [185](#)
 - HAL_CRY_ENFORCE_FALLBACK
 - Configuration, [331](#)
 - Crypto Driver, [77](#)
 - HAL_CRY_MAX_KEY_SIZE
 - Crypto Driver, [77](#)
 - HAL_CRY_USE_FALLBACK
 - Configuration, [330](#)
 - Crypto Driver, [77](#)
 - HAL_USE_ADC
 - Configuration, [328](#)
 - HAL_USE_CAN
 - Configuration, [328](#)
 - HAL_USE_CRY
 - Configuration, [328](#)
 - HAL_USE_DAC
 - Configuration, [328](#)
 - HAL_USE_EXT
 - Configuration, [328](#)
 - HAL_USE_GPT
 - Configuration, [328](#)
 - HAL_USE_I2C
 - Configuration, [328](#)
 - HAL_USE_I2S
 - Configuration, [329](#)
 - HAL_USE_ICU
 - Configuration, [329](#)
 - HAL_USE_MAC
 - Configuration, [329](#)
 - HAL_USE_MMC_SPI
 - Configuration, [329](#)
 - HAL_USE_PAL
 - Configuration, [328](#)
 - HAL_USE_PWM
 - Configuration, [329](#)
 - HAL_USE_QSPI
 - Configuration, [329](#)
 - HAL_USE_RTC
 - Configuration, [329](#)
 - HAL_USE_SDC
 - Configuration, [329](#)
 - HAL_USE_SERIAL_USB
 - Configuration, [329](#)
 - HAL_USE_SERIAL
 - Configuration, [329](#)
 - HAL_USE_SPI
 - Configuration, [329](#)
 - HAL_USE_UART
 - Configuration, [329](#)
 - HAL_USE_USB
 - Configuration, [330](#)

HAL_USE_WDG
 Configuration, 330
 HAL_VERSION
 HAL Driver, 185
 HAL, 323
 hal.c, 841
 hal.h, 841
 hal_accelerometer.h, 843
 hal_adc.c, 844
 hal_adc.h, 844
 hal_adc_llc.c, 846
 hal_adc_llc.h, 846
 hal_barometer.h, 847
 hal_buffers.c, 848
 hal_buffers.h, 849
 hal_can.c, 851
 hal_can.h, 852
 hal_can_llc.c, 854
 hal_can_llc.h, 854
 hal_channels.h, 855
 hal_compass.h, 857
 hal_crypto.c, 858
 hal_crypto.h, 860
 hal_crypto_llc.c, 862
 hal_crypto_llc.h, 864
 hal_dac.c, 866
 hal_dac.h, 867
 hal_dac_llc.c, 868
 hal_dac_llc.h, 869
 hal_ext.c, 870
 hal_ext.h, 871
 hal_ext_llc.c, 872
 hal_ext_llc.h, 873
 hal_files.h, 873
 hal_flash.c, 875
 hal_flash.h, 875
 hal_gpt.c, 876
 hal_gpt.h, 877
 hal_gpt_llc.c, 878
 hal_gpt_llc.h, 879
 hal_gyroscope.h, 880
 hal_hygrometer.h, 881
 hal_i2c.c, 882
 hal_i2c.h, 882
 hal_i2c_llc.c, 884
 hal_i2c_llc.h, 884
 hal_i2s.c, 885
 hal_i2s.h, 886
 hal_i2s_llc.c, 887
 hal_i2s_llc.h, 887
 hal_icu.c, 888
 hal_icu.h, 889
 hal_icu_llc.c, 890
 hal_icu_llc.h, 891
 hal_ioblock.h, 892
 hal_jesd216_flash.c, 893
 hal_jesd216_flash.h, 893
 hal_llc.c, 894
 hal_llc.h, 894
 hal_llc_init
 HAL Driver, 186
 hal_mac.c, 895
 hal_mac.h, 895
 hal_mac_llc.c, 897
 hal_mac_llc.h, 897
 hal_mii.h, 899
 hal_mmc_spi.c, 901
 hal_mmc_spi.h, 902
 hal_mmcsd.c, 903
 hal_mmcsd.h, 904
 hal_objects.h, 908
 hal_pal.c, 909
 hal_pal.h, 909
 hal_pal_llc.c, 913
 hal_pal_llc.h, 913
 hal_pwm.c, 915
 hal_pwm.h, 916
 hal_pwm_llc.c, 917
 hal_pwm_llc.h, 918
 hal_qspi.c, 919
 hal_qspi.h, 920
 hal_qspi_llc.c, 922
 hal_qspi_llc.h, 923
 hal_queues.c, 924
 hal_queues.h, 925
 hal_rtc.c, 927
 hal_rtc.h, 928
 hal_rtc_llc.c, 929
 hal_rtc_llc.h, 929
 hal_sdc.c, 930
 hal_sdc.h, 932
 hal_sdc_llc.c, 933
 hal_sdc_llc.h, 934
 hal_sensors.h, 936
 hal_serial.c, 936
 hal_serial.h, 937
 hal_serial_llc.c, 939
 hal_serial_llc.h, 939
 hal_serial_usb.c, 940
 hal_serial_usb.h, 941
 hal_spi.c, 942
 hal_spi.h, 943
 hal_spi_llc.c, 945
 hal_spi_llc.h, 946
 hal_st.c, 947
 hal_st.h, 948
 hal_st_llc.c, 948
 hal_st_llc.h, 949
 hal_streams.h, 949
 hal_thermometer.h, 950
 hal_uart.c, 951
 hal_uart.h, 952
 hal_uart_llc.c, 954
 hal_uart_llc.h, 954
 hal_usb.c, 956
 hal_usb.h, 957

- hal_usb_cdc.h, 960
- hal_usb_llc.c, 961
 - in, 962
 - out, 962
- hal_usb_llc.h, 962
- hal_wdg.c, 964
- hal_wdg.h, 964
- hal_wdg_llc.c, 965
- hal_wdg_llc.h, 965
- hallInit
 - HAL Driver, 185
- halconf.h, 966
- hscfg
 - MMCCConfig, 782
- hygrometerGetChannelsNumber
 - Generic Hygrometer Interface, 260
- hygrometerReadCooked
 - Generic Hygrometer Interface, 261
- hygrometerReadRaw
 - Generic Hygrometer Interface, 260
- hygrometerResetBias
 - Generic Hygrometer Interface, 262
- hygrometerResetSensitivity
 - Generic Hygrometer Interface, 263
- hygrometerSetBias
 - Generic Hygrometer Interface, 261
- hygrometerSetSensitivity
 - Generic Hygrometer Interface, 262
- I/O Buffers Queues, 187
 - BQ_BUFFER_SIZE, 189
 - bqGetLinkX, 189
 - bqIsSuspendedX, 190
 - bqResumeX, 191
 - bqSizeX, 189
 - bqSpacel, 189
 - bqSuspendI, 190
 - bqnotify_t, 193
 - ibqGetEmptyBufferI, 195
 - ibqGetFullBufferTimeout, 196
 - ibqGetFullBufferTimeoutS, 197
 - ibqGetTimeout, 199
 - ibqIsEmptyI, 191
 - ibqIsFullI, 191
 - ibqObjectInit, 194
 - ibqPostFullBufferI, 195
 - ibqReadTimeout, 199
 - ibqReleaseEmptyBuffer, 198
 - ibqReleaseEmptyBufferS, 198
 - ibqResetI, 194
 - input_buffers_queue_t, 193
 - io_buffers_queue_t, 193
 - obqFlush, 208
 - obqGetEmptyBufferTimeout, 203
 - obqGetEmptyBufferTimeoutS, 204
 - obqGetFullBufferI, 202
 - obqIsEmptyI, 192
 - obqIsFullI, 193
 - obqObjectInit, 200
 - obqPostFullBuffer, 205
 - obqPostFullBufferS, 205
 - obqPutTimeout, 206
 - obqReleaseEmptyBufferI, 202
 - obqResetI, 201
 - obqTryFlushI, 208
 - obqWriteTimeout, 207
 - output_buffers_queue_t, 193
- I/O Bytes Queues, 236
 - input_queue_t, 244
 - io_queue_t, 244
 - iq_read, 244
 - iqGet, 241
 - iqGetEmptyI, 240
 - iqGetFullI, 239
 - iqGetTimeout, 248
 - iqGetI, 247
 - iqIsEmptyI, 240
 - iqIsFullI, 241
 - iqObjectInit, 245
 - iqPutI, 246
 - iqReadTimeout, 249
 - iqReadI, 248
 - iqResetI, 246
 - oq_write, 245
 - oqGetEmptyI, 242
 - oqGetFullI, 242
 - oqGetI, 253
 - oqIsEmptyI, 242
 - oqIsFullI, 243
 - oqObjectInit, 250
 - oqPut, 243
 - oqPutTimeout, 252
 - oqPutI, 251
 - oqResetI, 251
 - oqWriteTimeout, 254
 - oqWriteI, 253
 - output_queue_t, 244
 - Q_EMPTY, 238
 - Q_FULL, 238
 - Q_OK, 238
 - Q_RESET, 238
 - Q_TIMEOUT, 238
 - qGetLink, 239
 - qSizeX, 238
 - qSpacel, 238
 - qnotify_t, 244
- I2C Driver, 264
 - _i2c_wakeup_error_isr, 267
 - _i2c_wakeup_isr, 267
 - I2C_ACK_FAILURE, 267
 - I2C_ACTIVE_RX, 269
 - I2C_ACTIVE_TX, 269
 - I2C_ARBITRATION_LOST, 266
 - I2C_BUS_ERROR, 266
 - I2C_NO_ERROR, 266
 - I2C_OVERRUN, 267
 - I2C_PEC_ERROR, 267

- I2C_READY, [269](#)
- I2C_SMB_ALERT, [267](#)
- I2C_STOP, [269](#)
- I2C_TIMEOUT, [267](#)
- I2C_UNINIT, [269](#)
- I2C_USE_MUTUAL_EXCLUSION, [267](#)
- I2CD1, [277](#)
- I2CDriver, [269](#)
- i2c_llid_get_errors, [268](#)
- i2c_llid_init, [275](#)
- i2c_llid_master_receive_timeout, [276](#)
- i2c_llid_master_transmit_timeout, [276](#)
- i2c_llid_start, [275](#)
- i2c_llid_stop, [276](#)
- i2cAcquireBus, [274](#)
- i2cGetErrors, [271](#)
- i2cInit, [269](#)
- i2cMasterReceive, [268](#)
- i2cMasterReceiveTimeout, [273](#)
- i2cMasterTransmit, [268](#)
- i2cMasterTransmitTimeout, [272](#)
- i2cObjectInit, [270](#)
- i2cReleaseBus, [274](#)
- i2cStart, [270](#)
- i2cStop, [271](#)
- i2caddr_t, [269](#)
- i2cflags_t, [269](#)
- i2cstate_t, [269](#)
- PLATFORM_I2C_USE_I2C1, [268](#)
- I2C_ACK_FAILURE
 - I2C Driver, [267](#)
- I2C_ACTIVE_RX
 - I2C Driver, [269](#)
- I2C_ACTIVE_TX
 - I2C Driver, [269](#)
- I2C_ARBITRATION_LOST
 - I2C Driver, [266](#)
- I2C_BUS_ERROR
 - I2C Driver, [266](#)
- I2C_NO_ERROR
 - I2C Driver, [266](#)
- I2C_OVERRUN
 - I2C Driver, [267](#)
- I2C_PEC_ERROR
 - I2C Driver, [267](#)
- I2C_READY
 - I2C Driver, [269](#)
- I2C_SMB_ALERT
 - I2C Driver, [267](#)
- I2C_STOP
 - I2C Driver, [269](#)
- I2C_TIMEOUT
 - I2C Driver, [267](#)
- I2C_UNINIT
 - I2C Driver, [269](#)
- I2C_USE_MUTUAL_EXCLUSION
 - Configuration, [331](#)
 - I2C Driver, [267](#)
- I2CConfig, [756](#)
- I2CD1
 - I2C Driver, [277](#)
- I2CDriver, [757](#)
 - config, [758](#)
 - errors, [758](#)
 - I2C Driver, [269](#)
 - state, [758](#)
- I2S Driver, [278](#)
 - _i2s_isr_full_code, [280](#)
 - _i2s_isr_half_code, [280](#)
 - I2S_ACTIVE, [282](#)
 - I2S_COMPLETE, [282](#)
 - I2S_READY, [282](#)
 - I2S_STOP, [282](#)
 - I2S_UNINIT, [282](#)
 - I2SD1, [286](#)
 - I2SDriver, [281](#)
 - i2s_llid_init, [285](#)
 - i2s_llid_start, [285](#)
 - i2s_llid_start_exchange, [286](#)
 - i2s_llid_stop, [285](#)
 - i2s_llid_stop_exchange, [286](#)
 - i2sInit, [282](#)
 - i2sObjectInit, [282](#)
 - i2sStart, [282](#)
 - i2sStartExchange, [284](#)
 - i2sStartExchangeI, [279](#)
 - i2sStop, [283](#)
 - i2sStopExchange, [284](#)
 - i2sStopExchangeI, [279](#)
 - i2scallback_t, [281](#)
 - i2sstate_t, [282](#)
 - PLATFORM_I2S_USE_I2S1, [281](#)
- I2S_ACTIVE
 - I2S Driver, [282](#)
- I2S_COMPLETE
 - I2S Driver, [282](#)
- I2S_READY
 - I2S Driver, [282](#)
- I2S_STOP
 - I2S Driver, [282](#)
- I2S_UNINIT
 - I2S Driver, [282](#)
- I2SConfig, [758](#)
 - end_cb, [759](#)
 - rx_buffer, [759](#)
 - size, [759](#)
 - tx_buffer, [759](#)
- I2SD1
 - I2S Driver, [286](#)
- I2SDriver, [759](#)
 - config, [760](#)
 - I2S Driver, [281](#)
 - state, [760](#)
- i2c_llid_get_errors
 - I2C Driver, [268](#)
- i2c_llid_init

- I2C Driver, [275](#)
- i2c_llid_master_receive_timeout
 - I2C Driver, [276](#)
- i2c_llid_master_transmit_timeout
 - I2C Driver, [276](#)
- i2c_llid_start
 - I2C Driver, [275](#)
- i2c_llid_stop
 - I2C Driver, [276](#)
- i2cAcquireBus
 - I2C Driver, [274](#)
- i2cGetErrors
 - I2C Driver, [271](#)
- i2cInit
 - I2C Driver, [269](#)
- i2cMasterReceive
 - I2C Driver, [268](#)
- i2cMasterReceiveTimeout
 - I2C Driver, [273](#)
- i2cMasterTransmit
 - I2C Driver, [268](#)
- i2cMasterTransmitTimeout
 - I2C Driver, [272](#)
- i2cObjectInit
 - I2C Driver, [270](#)
- i2cReleaseBus
 - I2C Driver, [274](#)
- i2cStart
 - I2C Driver, [270](#)
- i2cStop
 - I2C Driver, [271](#)
- i2caddr_t
 - I2C Driver, [269](#)
- i2cflags_t
 - I2C Driver, [269](#)
- i2cstate_t
 - I2C Driver, [269](#)
- i2s_llid_init
 - I2S Driver, [285](#)
- i2s_llid_start
 - I2S Driver, [285](#)
- i2s_llid_start_exchange
 - I2S Driver, [286](#)
- i2s_llid_stop
 - I2S Driver, [285](#)
- i2s_llid_stop_exchange
 - I2S Driver, [286](#)
- i2sInit
 - I2S Driver, [282](#)
- i2sObjectInit
 - I2S Driver, [282](#)
- i2sStart
 - I2S Driver, [282](#)
- i2sStartExchange
 - I2S Driver, [284](#)
- i2sStartExchange1
 - I2S Driver, [279](#)
- i2sStop
 - I2S Driver, [283](#)
- i2sStopExchange
 - I2S Driver, [284](#)
- i2sStopExchange1
 - I2S Driver, [279](#)
- i2scallback_t
 - I2S Driver, [281](#)
- i2sstate_t
 - I2S Driver, [282](#)
- ICU Driver, [287](#)
 - _icu_isr_invoke_overflow_cb, [293](#)
 - _icu_isr_invoke_period_cb, [293](#)
 - _icu_isr_invoke_width_cb, [293](#)
 - ICU_ACTIVE, [296](#)
 - ICU_INPUT_ACTIVE_HIGH, [296](#)
 - ICU_INPUT_ACTIVE_LOW, [296](#)
 - ICU_READY, [296](#)
 - ICU_STOP, [296](#)
 - ICU_UNINIT, [296](#)
 - ICU_WAITING, [296](#)
 - ICUD1, [304](#)
 - ICUDriver, [295](#)
 - icu_llid_are_notifications_enabled, [295](#)
 - icu_llid_disable_notifications, [303](#)
 - icu_llid_enable_notifications, [303](#)
 - icu_llid_get_period, [294](#)
 - icu_llid_get_width, [294](#)
 - icu_llid_init, [301](#)
 - icu_llid_start, [301](#)
 - icu_llid_start_capture, [302](#)
 - icu_llid_stop, [302](#)
 - icu_llid_stop_capture, [303](#)
 - icu_llid_wait_capture, [302](#)
 - icuAreNotificationsEnabledX, [291](#)
 - icuDisableNotifications, [300](#)
 - icuDisableNotifications1, [291](#)
 - icuEnableNotifications, [300](#)
 - icuEnableNotifications1, [290](#)
 - icuGetPeriodX, [292](#)
 - icuGetWidthX, [292](#)
 - icuInit, [296](#)
 - icuObjectInit, [296](#)
 - icuStart, [297](#)
 - icuStartCapture, [298](#)
 - icuStartCapture1, [290](#)
 - icuStop, [297](#)
 - icuStopCapture, [299](#)
 - icuStopCapture1, [290](#)
 - icuWaitCapture, [298](#)
 - icucallback_t, [295](#)
 - icucnt_t, [295](#)
 - icufreq_t, [295](#)
 - icumode_t, [296](#)
 - icustate_t, [296](#)
 - PLATFORM_ICU_USE_ICU1, [294](#)
- ICU_ACTIVE
 - ICU Driver, [296](#)
- ICU_INPUT_ACTIVE_HIGH

- ICU Driver, [296](#)
- ICU_INPUT_ACTIVE_LOW
 - ICU Driver, [296](#)
- ICU_READY
 - ICU Driver, [296](#)
- ICU_STOP
 - ICU Driver, [296](#)
- ICU_UNINIT
 - ICU Driver, [296](#)
- ICU_WAITING
 - ICU Driver, [296](#)
- ICUConfig, [760](#)
 - frequency, [761](#)
 - mode, [761](#)
 - overflow_cb, [762](#)
 - period_cb, [762](#)
 - width_cb, [762](#)
- ICUD1
 - ICU Driver, [304](#)
- ICUDriver, [762](#)
 - config, [763](#)
 - ICU Driver, [295](#)
 - state, [763](#)
- IDE
 - CANRxFrame, [736](#)
 - CANTxFrame, [738](#)
- IOBUS_DECL
 - PAL Driver, [433](#)
- IOBus, [767](#)
 - mask, [768](#)
 - offset, [768](#)
 - portid, [768](#)
- IOPORT1
 - PAL Driver, [450](#)
- ibnotify
 - Serial over USB Driver, [568](#)
- ibqGetEmptyBufferI
 - I/O Buffers Queues, [195](#)
- ibqGetFullBufferTimeout
 - I/O Buffers Queues, [196](#)
- ibqGetFullBufferTimeoutS
 - I/O Buffers Queues, [197](#)
- ibqGetTimeout
 - I/O Buffers Queues, [199](#)
- ibqlsEmptyI
 - I/O Buffers Queues, [191](#)
- ibqlsFullI
 - I/O Buffers Queues, [191](#)
- ibqObjectInit
 - I/O Buffers Queues, [194](#)
- ibqPostFullBufferI
 - I/O Buffers Queues, [195](#)
- ibqReadTimeout
 - I/O Buffers Queues, [199](#)
- ibqReleaseEmptyBuffer
 - I/O Buffers Queues, [198](#)
- ibqReleaseEmptyBufferS
 - I/O Buffers Queues, [198](#)
- ibqResetI
 - I/O Buffers Queues, [194](#)
- icu_lld_are_notifications_enabled
 - ICU Driver, [295](#)
- icu_lld_disable_notifications
 - ICU Driver, [303](#)
- icu_lld_enable_notifications
 - ICU Driver, [303](#)
- icu_lld_get_period
 - ICU Driver, [294](#)
- icu_lld_get_width
 - ICU Driver, [294](#)
- icu_lld_init
 - ICU Driver, [301](#)
- icu_lld_start
 - ICU Driver, [301](#)
- icu_lld_start_capture
 - ICU Driver, [302](#)
- icu_lld_stop
 - ICU Driver, [302](#)
- icu_lld_stop_capture
 - ICU Driver, [303](#)
- icu_lld_wait_capture
 - ICU Driver, [302](#)
- icuAreNotificationsEnabledX
 - ICU Driver, [291](#)
- icuDisableNotifications
 - ICU Driver, [300](#)
- icuDisableNotificationsI
 - ICU Driver, [291](#)
- icuEnableNotifications
 - ICU Driver, [300](#)
- icuEnableNotificationsI
 - ICU Driver, [290](#)
- icuGetPeriodX
 - ICU Driver, [292](#)
- icuGetWidthX
 - ICU Driver, [292](#)
- iculnit
 - ICU Driver, [296](#)
- icuObjectInit
 - ICU Driver, [296](#)
- icuStart
 - ICU Driver, [297](#)
- icuStartCapture
 - ICU Driver, [298](#)
- icuStartCaptureI
 - ICU Driver, [290](#)
- icuStop
 - ICU Driver, [297](#)
- icuStopCapture
 - ICU Driver, [299](#)
- icuStopCaptureI
 - ICU Driver, [290](#)
- icuWaitCapture
 - ICU Driver, [298](#)
- icucallback_t
 - ICU Driver, [295](#)

- icucnt_t
 - ICU Driver, [295](#)
- icufreq_t
 - ICU Driver, [295](#)
- icumode_t
 - ICU Driver, [296](#)
- icustate_t
 - ICU Driver, [296](#)
- id
 - mfs_data_header_t, [775](#)
- in
 - hal_usb_llid.c, [962](#)
 - USB Driver, [682](#)
- in_cb
 - USBEndpointConfig, [835](#)
- in_maxsize
 - USBEndpointConfig, [835](#)
- in_params
 - USBDriver, [832](#)
- in_state
 - USBEndpointConfig, [836](#)
- Inner Code, [339](#)
- input_buffers_queue_t
 - I/O Buffers Queues, [193](#)
- input_queue_t
 - I/O Bytes Queues, [244](#)
- int_in
 - SerialUSBConfig, [811](#)
- Interfaces and Classes, [338](#)
- io_buffers_queue, [763](#)
 - bcounter, [764](#)
 - bn, [765](#)
 - brdptr, [764](#)
 - bsize, [765](#)
 - btop, [765](#)
 - buffers, [765](#)
 - bwrptr, [764](#)
 - link, [765](#)
 - notify, [765](#)
 - ptr, [765](#)
 - suspended, [764](#)
 - top, [765](#)
 - waiting, [764](#)
- io_buffers_queue_t
 - I/O Buffers Queues, [193](#)
- io_queue, [765](#)
 - q_buffer, [767](#)
 - q_counter, [767](#)
 - q_link, [767](#)
 - q_notify, [767](#)
 - q_rdptra, [767](#)
 - q_top, [767](#)
 - q_waiting, [767](#)
 - q_wrptra, [767](#)
- io_queue_t
 - I/O Bytes Queues, [244](#)
- ioline_t
 - PAL Driver, [458](#)
- iomode_t
 - PAL Driver, [458](#)
- iopadid_t
 - PAL Driver, [458](#)
- ioportid_t
 - PAL Driver, [458](#)
- ioportmask_t
 - PAL Driver, [458](#)
- iq_read
 - I/O Bytes Queues, [244](#)
- iqGet
 - I/O Bytes Queues, [241](#)
- iqGetEmptyI
 - I/O Bytes Queues, [240](#)
- iqGetFullI
 - I/O Bytes Queues, [239](#)
- iqGetTimeout
 - I/O Bytes Queues, [248](#)
- iqGetI
 - I/O Bytes Queues, [247](#)
- iqIsEmptyI
 - I/O Bytes Queues, [240](#)
- iqIsFullI
 - I/O Bytes Queues, [241](#)
- iqObjectInit
 - I/O Bytes Queues, [245](#)
- iqPutI
 - I/O Bytes Queues, [246](#)
- iqReadTimeout
 - I/O Bytes Queues, [249](#)
- iqReadI
 - I/O Bytes Queues, [248](#)
- iqResetI
 - I/O Bytes Queues, [246](#)
- JESD215Flash, [769](#)
 - vmt, [769](#)
- JESD215FlashVMT, [769](#)
- JESD216 Compliant Flash Driver Interface, [305](#)
 - _jesd216_flash_data, [306](#)
 - _jesd216_flash_methods, [306](#)
 - _jesd216_flash_methods_alone, [306](#)
 - JESD216_BUS_MODE, [306](#)
 - JESD216_SHARED_BUS, [306](#)
- JESD216_BUS_MODE
 - JESD216 Compliant Flash Driver Interface, [306](#)
- JESD216_SHARED_BUS
 - JESD216 Compliant Flash Driver Interface, [306](#)
- key0_buffer
 - CRYDriver, [741](#)
- key0_size
 - CRYDriver, [741](#)
- key0_type
 - CRYDriver, [740](#)
- LPA_100BASE4
 - MII/RMII Header, [390](#)
- LPA_100FULL

- MII/RMII Header, [390](#)
- LPA_100HALF
 - MII/RMII Header, [390](#)
- LPA_10FULL
 - MII/RMII Header, [390](#)
- LPA_10HALF
 - MII/RMII Header, [390](#)
- LPA_LPACK
 - MII/RMII Header, [391](#)
- LPA_NPAGE
 - MII/RMII Header, [391](#)
- LPA_PAUSE_ASYM
 - MII/RMII Header, [391](#)
- LPA_PAUSE_CAP
 - MII/RMII Header, [391](#)
- LPA_RESV
 - MII/RMII Header, [391](#)
- LPA_RFAULT
 - MII/RMII Header, [391](#)
- LPA_SLCT
 - MII/RMII Header, [390](#)
- link
 - io_buffers_queue, [765](#)
- lscfg
 - MMCCConfig, [782](#)
- MAC Driver, [307](#)
 - ETHD1, [322](#)
 - MAC_ACTIVE, [312](#)
 - MAC_STOP, [312](#)
 - MAC_SUPPORTS_ZERO_COPY, [311](#)
 - MAC_UNINIT, [312](#)
 - MAC_USE_EVENTS, [309](#)
 - MAC_USE_ZERO_COPY, [309](#)
 - MACDriver, [311](#)
 - mac_lld_get_next_receive_buffer, [321](#)
 - mac_lld_get_next_transmit_buffer, [321](#)
 - mac_lld_get_receive_descriptor, [319](#)
 - mac_lld_get_transmit_descriptor, [318](#)
 - mac_lld_init, [317](#)
 - mac_lld_poll_link_status, [320](#)
 - mac_lld_read_receive_descriptor, [320](#)
 - mac_lld_release_receive_descriptor, [319](#)
 - mac_lld_release_transmit_descriptor, [319](#)
 - mac_lld_start, [318](#)
 - mac_lld_stop, [318](#)
 - mac_lld_write_transmit_descriptor, [320](#)
 - macGetNextReceiveBuffer, [310](#)
 - macGetNextTransmitBuffer, [310](#)
 - macGetReceiveEventSource, [309](#)
 - macInit, [312](#)
 - macObjectInit, [312](#)
 - macPollLinkStatus, [317](#)
 - macReadReceiveDescriptor, [310](#)
 - macReleaseReceiveDescriptor, [316](#)
 - macReleaseTransmitDescriptor, [315](#)
 - macStart, [313](#)
 - macStop, [313](#)
 - macWaitReceiveDescriptor, [315](#)
 - macWaitTransmitDescriptor, [314](#)
 - macWriteTransmitDescriptor, [309](#)
 - macstate_t, [311](#)
 - PLATFORM_MAC_USE_MAC1, [311](#)
- MAC_ACTIVE
 - MAC Driver, [312](#)
- MAC_STOP
 - MAC Driver, [312](#)
- MAC_SUPPORTS_ZERO_COPY
 - MAC Driver, [311](#)
- MAC_UNINIT
 - MAC Driver, [312](#)
- MAC_USE_EVENTS
 - Configuration, [331](#)
 - MAC Driver, [309](#)
- MAC_USE_ZERO_COPY
 - Configuration, [331](#)
 - MAC Driver, [309](#)
- MACConfig, [770](#)
 - mac_address, [770](#)
- MACDriver, [771](#)
 - config, [771](#)
 - MAC Driver, [311](#)
 - rdevent, [772](#)
 - rdqueue, [772](#)
 - state, [771](#)
 - tdqueue, [772](#)
- MACReceiveDescriptor, [772](#)
 - offset, [773](#)
 - size, [773](#)
- MACTransmitDescriptor, [773](#)
 - offset, [773](#)
 - size, [773](#)
- MFS_CFG_BUFFER_SIZE
 - Managed Flash Storage Driver, [367](#)
- MFS_CFG_MAX_RECORDS
 - Managed Flash Storage Driver, [367](#)
- MFS_CFG_MAX_REPAIR_ATTEMPTS
 - Managed Flash Storage Driver, [367](#)
- MFS_CFG_STRONG_CHECKING
 - Managed Flash Storage Driver, [367](#)
- MFS_CFG_WRITE_VERIFY
 - Managed Flash Storage Driver, [367](#)
- MFSCConfig, [776](#)
 - bank0_sectors, [777](#)
 - bank0_start, [777](#)
 - bank1_sectors, [777](#)
 - bank1_start, [777](#)
 - bank_size, [777](#)
 - erased, [777](#)
 - flashp, [777](#)
- MFSDriver, [777](#)
 - buffer, [780](#)
 - config, [780](#)
 - current_bank, [780](#)
 - current_counter, [780](#)
 - descriptors, [780](#)
 - next_offset, [780](#)

- state, [780](#)
- used_space, [780](#)
- MII/RMII Header, [383](#)
 - ADVERTISE_100BASE4, [389](#)
 - ADVERTISE_100FULL, [389](#)
 - ADVERTISE_100HALF, [389](#)
 - ADVERTISE_10FULL, [389](#)
 - ADVERTISE_10HALF, [389](#)
 - ADVERTISE_CSMA, [389](#)
 - ADVERTISE_LPACK, [390](#)
 - ADVERTISE_NPAGE, [390](#)
 - ADVERTISE_PAUSE_ASYM, [390](#)
 - ADVERTISE_PAUSE_CAP, [390](#)
 - ADVERTISE_RESV, [390](#)
 - ADVERTISE_RFAULT, [390](#)
 - ADVERTISE_SLCT, [389](#)
 - BMCR_ANENABLE, [388](#)
 - BMCR_ANRESTART, [387](#)
 - BMCR_CTST, [387](#)
 - BMCR_FULLDPLX, [387](#)
 - BMCR_ISOLATE, [387](#)
 - BMCR_LOOPBACK, [388](#)
 - BMCR_PDOWN, [387](#)
 - BMCR_RESET, [388](#)
 - BMCR_RESV, [387](#)
 - BMCR_SPEED100, [388](#)
 - BMSR_100BASE4, [389](#)
 - BMSR_100FULL, [389](#)
 - BMSR_100HALF, [389](#)
 - BMSR_10FULL, [389](#)
 - BMSR_10HALF, [389](#)
 - BMSR_ANEGCAPABLE, [388](#)
 - BMSR_ANEGCOMPLETE, [388](#)
 - BMSR_ERCAP, [388](#)
 - BMSR_JCD, [388](#)
 - BMSR_LSTATUS, [388](#)
 - BMSR_MFPRESUPPCAP, [388](#)
 - BMSR_RESV, [388](#)
 - BMSR_RFAULT, [388](#)
 - EXPANSION_ENABLENPAGE, [391](#)
 - EXPANSION_LCWP, [391](#)
 - EXPANSION_MFAULTS, [391](#)
 - EXPANSION_NPCAPABLE, [391](#)
 - EXPANSION_NWAY, [391](#)
 - EXPANSION_RESV, [391](#)
 - LPA_100BASE4, [390](#)
 - LPA_100FULL, [390](#)
 - LPA_100HALF, [390](#)
 - LPA_10FULL, [390](#)
 - LPA_10HALF, [390](#)
 - LPA_LPACK, [391](#)
 - LPA_NPAGE, [391](#)
 - LPA_PAUSE_ASYM, [391](#)
 - LPA_PAUSE_CAP, [391](#)
 - LPA_RESV, [391](#)
 - LPA_RFAULT, [391](#)
 - LPA_SLCT, [390](#)
 - MII_ADVERTISE, [385](#)
 - MII_ANNPTR, [386](#)
 - MII_BMCR, [385](#)
 - MII_BMSR, [385](#)
 - MII_CTRL1000, [386](#)
 - MII_DCOUNTER, [386](#)
 - MII_ESTATUS, [386](#)
 - MII_EXPANSION, [386](#)
 - MII_FCSCOUNTER, [386](#)
 - MII_LBRERROR, [387](#)
 - MII_LPA, [385](#)
 - MII_MICR, [386](#)
 - MII_NCONFIG, [387](#)
 - MII_NWAYTEST, [386](#)
 - MII_PHYADDR, [387](#)
 - MII_PHYSID1, [385](#)
 - MII_PHYSID2, [385](#)
 - MII_PHYSTS, [386](#)
 - MII_RERRCOUNTER, [386](#)
 - MII_RESV1, [387](#)
 - MII_RESV2, [387](#)
 - MII_SREVISION, [386](#)
 - MII_STAT1000, [386](#)
 - MII_TPISTATUS, [387](#)
 - NWAYTEST_LOOPBACK, [392](#)
 - NWAYTEST_RESV1, [392](#)
 - NWAYTEST_RESV2, [392](#)
- MII_ADVERTISE
 - MII/RMII Header, [385](#)
- MII_ANNPTR
 - MII/RMII Header, [386](#)
- MII_BMCR
 - MII/RMII Header, [385](#)
- MII_BMSR
 - MII/RMII Header, [385](#)
- MII_CTRL1000
 - MII/RMII Header, [386](#)
- MII_DCOUNTER
 - MII/RMII Header, [386](#)
- MII_ESTATUS
 - MII/RMII Header, [386](#)
- MII_EXPANSION
 - MII/RMII Header, [386](#)
- MII_FCSCOUNTER
 - MII/RMII Header, [386](#)
- MII_LBRERROR
 - MII/RMII Header, [387](#)
- MII_LPA
 - MII/RMII Header, [385](#)
- MII_MICR
 - MII/RMII Header, [386](#)
- MII_NCONFIG
 - MII/RMII Header, [387](#)
- MII_NWAYTEST
 - MII/RMII Header, [386](#)
- MII_PHYADDR
 - MII/RMII Header, [387](#)
- MII_PHYSID1
 - MII/RMII Header, [385](#)

- MII_PHYSID2
 - MII/RMII Header, [385](#)
- MII_PHYSTS
 - MII/RMII Header, [386](#)
- MII_RERRCOUNTER
 - MII/RMII Header, [386](#)
- MII_RESV1
 - MII/RMII Header, [387](#)
- MII_RESV2
 - MII/RMII Header, [387](#)
- MII_SREVISION
 - MII/RMII Header, [386](#)
- MII_STAT1000
 - MII/RMII Header, [386](#)
- MII_TPISTATUS
 - MII/RMII Header, [387](#)
- MMC over SPI Driver, [393](#)
 - _mmc_driver_methods, [395](#)
 - crc7, [396](#)
 - crc7_lookup_table, [412](#)
 - MMC_NICE_WAITING, [395](#)
 - mmc_vmt, [412](#)
 - mmcConnect, [403](#)
 - mmcDisconnect, [404](#)
 - mmcErase, [411](#)
 - mmcGetInfo, [411](#)
 - mmcInit, [402](#)
 - mmclsCardInserted, [395](#)
 - mmclsWriteProtected, [395](#)
 - mmcObjectInit, [402](#)
 - mmcSequentialRead, [406](#)
 - mmcSequentialWrite, [408](#)
 - mmcStart, [402](#)
 - mmcStartSequentialRead, [405](#)
 - mmcStartSequentialWrite, [407](#)
 - mmcStop, [403](#)
 - mmcStopSequentialRead, [407](#)
 - mmcStopSequentialWrite, [409](#)
 - mmcSync, [410](#)
 - read_CxD, [400](#)
 - recvr1, [397](#)
 - recvr3, [398](#)
 - send_command_R1, [399](#)
 - send_command_R3, [400](#)
 - send_hdr, [397](#)
 - sync, [401](#)
 - wait, [396](#)
- MMC/SD Block Device, [414](#)
 - _mmcscd_block_device_data, [419](#)
 - _mmcscd_block_device_methods, [418](#)
 - _mmcscd_get_capacity, [420](#)
 - _mmcscd_get_capacity_ext, [421](#)
 - _mmcscd_get_slice, [420](#)
 - _mmcscd_unpack_csd_mmc, [422](#)
 - _mmcscd_unpack_csd_v10, [423](#)
 - _mmcscd_unpack_csd_v20, [423](#)
 - _mmcscd_unpack_mmc_cid, [422](#)
 - _mmcscd_unpack_sdc_cid, [421](#)
 - MMCSDBLOCK_SIZE, [418](#)
 - MMCSDBLOCK_CID_SDC_CRC_SLICE, [418](#)
 - MMCSDBLOCK_CMD8_PATTERN, [418](#)
 - MMCSDBLOCK_CSD_MMC_CSD_STRUCTURE_SLICE, [418](#)
 - MMCSDBLOCK_R1_ERROR_MASK, [418](#)
 - MMCSDBLOCK_R1_ERROR, [419](#)
 - MMCSDBLOCK_R1_IS_CARD_LOCKED, [419](#)
 - MMCSDBLOCK_R1_STS, [419](#)
 - mmcscdGetCardCapacity, [419](#)
- MMC_NICE_WAITING
 - Configuration, [331](#)
 - MMC over SPI Driver, [395](#)
- MMCConfig, [781](#)
 - hscfg, [782](#)
 - lscfg, [782](#)
 - spip, [782](#)
- MMCDriver, [782](#)
 - config, [783](#)
 - vmt, [783](#)
- MMCDriverVMT, [784](#)
- MMCSDBLOCK_SIZE
 - MMC/SD Block Device, [418](#)
- MMCSDBLOCK_CID_SDC_CRC_SLICE
 - MMC/SD Block Device, [418](#)
- MMCSDBLOCK_CMD8_PATTERN
 - MMC/SD Block Device, [418](#)
- MMCSDBLOCK_CSD_MMC_CSD_STRUCTURE_SLICE
 - MMC/SD Block Device, [418](#)
- MMCSDBLOCK_R1_ERROR_MASK
 - MMC/SD Block Device, [418](#)
- MMCSDBLOCK_R1_ERROR
 - MMC/SD Block Device, [419](#)
- MMCSDBLOCK_R1_IS_CARD_LOCKED
 - MMC/SD Block Device, [419](#)
- MMCSDBLOCK_R1_STS
 - MMC/SD Block Device, [419](#)
- MMCSDBlockDevice, [785](#)
 - vmt, [787](#)
- MMCSDBlockDeviceVMT, [787](#)
- mac_address
 - MACConfig, [770](#)
- mac_lld_get_next_receive_buffer
 - MAC Driver, [321](#)
- mac_lld_get_next_transmit_buffer
 - MAC Driver, [321](#)
- mac_lld_get_receive_descriptor
 - MAC Driver, [319](#)
- mac_lld_get_transmit_descriptor
 - MAC Driver, [318](#)
- mac_lld_init
 - MAC Driver, [317](#)
- mac_lld_poll_link_status
 - MAC Driver, [320](#)
- mac_lld_read_receive_descriptor
 - MAC Driver, [320](#)
- mac_lld_release_receive_descriptor
 - MAC Driver, [319](#)

- mac_lld_release_transmit_descriptor
 - MAC Driver, [319](#)
- mac_lld_start
 - MAC Driver, [318](#)
- mac_lld_stop
 - MAC Driver, [318](#)
- mac_lld_write_transmit_descriptor
 - MAC Driver, [320](#)
- macGetNextReceiveBuffer
 - MAC Driver, [310](#)
- macGetNextTransmitBuffer
 - MAC Driver, [310](#)
- macGetReceiveEventSource
 - MAC Driver, [309](#)
- macInit
 - MAC Driver, [312](#)
- macObjectInit
 - MAC Driver, [312](#)
- macPollLinkStatus
 - MAC Driver, [317](#)
- macReadReceiveDescriptor
 - MAC Driver, [310](#)
- macReleaseReceiveDescriptor
 - MAC Driver, [316](#)
- macReleaseTransmitDescriptor
 - MAC Driver, [315](#)
- macStart
 - MAC Driver, [313](#)
- macStop
 - MAC Driver, [313](#)
- macWaitReceiveDescriptor
 - MAC Driver, [315](#)
- macWaitTransmitDescriptor
 - MAC Driver, [314](#)
- macWriteTransmitDescriptor
 - MAC Driver, [309](#)
- macstate_t
 - MAC Driver, [311](#)
- magic
 - mfs_data_header_t, [775](#)
- magic1
 - mfs_bank_header_t, [774](#)
- magic2
 - mfs_bank_header_t, [774](#)
- Managed Flash Storage Driver, [365](#)
 - MFS_CFG_BUFFER_SIZE, [367](#)
 - MFS_CFG_MAX_RECORDS, [367](#)
 - MFS_CFG_MAX_REPAIR_ATTEMPTS, [367](#)
 - MFS_CFG_STRONG_CHECKING, [367](#)
 - MFS_CFG_WRITE_VERIFY, [367](#)
 - mfs_bank_erase, [370](#)
 - mfs_bank_get_state, [373](#)
 - mfs_bank_mount, [374](#)
 - mfs_bank_scan_records, [372](#)
 - mfs_bank_state_t, [368](#)
 - mfs_bank_t, [368](#)
 - mfs_bank_verify_erase, [371](#)
 - mfs_bank_write_header, [371](#)
- mfs_error_t, [368](#)
- mfs_flash_copy, [369](#)
- mfs_flash_read, [368](#)
- mfs_flash_write, [369](#)
- mfs_garbage_collect, [375](#)
- mfs_id_t, [368](#)
- mfs_mount, [376](#)
- mfs_record_check, [370](#)
- mfs_record_state_t, [368](#)
- mfs_state_t, [368](#)
- mfs_try_mount, [375](#)
- mfsErase, [378](#)
- mfsEraseRecord, [380](#)
- mfsObjectInit, [377](#)
- mfsPerformGarbageCollection, [381](#)
- mfsReadRecord, [379](#)
- mfsStart, [377](#)
- mfsStop, [378](#)
- mfsWriteRecord, [379](#)
- RET_ON_ERROR, [367](#)
- mask
 - IOBus, [768](#)
- mfs.c, [969](#)
- mfs.h, [970](#)
- mfs_bank_erase
 - Managed Flash Storage Driver, [370](#)
- mfs_bank_get_state
 - Managed Flash Storage Driver, [373](#)
- mfs_bank_header_t, [774](#)
 - counter, [774](#)
 - crc, [774](#)
 - magic1, [774](#)
 - magic2, [774](#)
 - reserved1, [774](#)
- mfs_bank_mount
 - Managed Flash Storage Driver, [374](#)
- mfs_bank_scan_records
 - Managed Flash Storage Driver, [372](#)
- mfs_bank_state_t
 - Managed Flash Storage Driver, [368](#)
- mfs_bank_t
 - Managed Flash Storage Driver, [368](#)
- mfs_bank_verify_erase
 - Managed Flash Storage Driver, [371](#)
- mfs_bank_write_header
 - Managed Flash Storage Driver, [371](#)
- mfs_data_header_t, [775](#)
 - crc, [775](#)
 - id, [775](#)
 - magic, [775](#)
 - size, [775](#)
- mfs_error_t
 - Managed Flash Storage Driver, [368](#)
- mfs_flash_copy
 - Managed Flash Storage Driver, [369](#)
- mfs_flash_read
 - Managed Flash Storage Driver, [368](#)
- mfs_flash_write

- Managed Flash Storage Driver, [369](#)
- mfs_garbage_collect
 - Managed Flash Storage Driver, [375](#)
- mfs_id_t
 - Managed Flash Storage Driver, [368](#)
- mfs_mount
 - Managed Flash Storage Driver, [376](#)
- mfs_record_check
 - Managed Flash Storage Driver, [370](#)
- mfs_record_state_t
 - Managed Flash Storage Driver, [368](#)
- mfs_state_t
 - Managed Flash Storage Driver, [368](#)
- mfs_try_mount
 - Managed Flash Storage Driver, [375](#)
- mfsErase
 - Managed Flash Storage Driver, [378](#)
- mfsEraseRecord
 - Managed Flash Storage Driver, [380](#)
- mfsObjectInit
 - Managed Flash Storage Driver, [377](#)
- mfsPerformGarbageCollection
 - Managed Flash Storage Driver, [381](#)
- mfsReadRecord
 - Managed Flash Storage Driver, [379](#)
- mfsStart
 - Managed Flash Storage Driver, [377](#)
- mfsStop
 - Managed Flash Storage Driver, [378](#)
- mfsWriteRecord
 - Managed Flash Storage Driver, [379](#)
- millisecond
 - RTCDatetime, [799](#)
- mmc_cmd6_construct
 - SDC Driver, [526](#)
- mmc_detect_bus_clk
 - SDC Driver, [528](#)
- mmc_init
 - SDC Driver, [525](#)
- mmc_set_bus_width
 - SDC Driver, [531](#)
- mmc_switch_t
 - SDC Driver, [523](#)
- mmc_vmt
 - MMC over SPI Driver, [412](#)
- mmcConnect
 - MMC over SPI Driver, [403](#)
- mmcDisconnect
 - MMC over SPI Driver, [404](#)
- mmcErase
 - MMC over SPI Driver, [411](#)
- mmcGetInfo
 - MMC over SPI Driver, [411](#)
- mmcInit
 - MMC over SPI Driver, [402](#)
- mmclsCardInserted
 - MMC over SPI Driver, [395](#)
- mmclsWriteProtected
 - MMC over SPI Driver, [395](#)
- mmcObjectInit
 - MMC over SPI Driver, [402](#)
- mmcSequentialRead
 - MMC over SPI Driver, [406](#)
- mmcSequentialWrite
 - MMC over SPI Driver, [408](#)
- mmcStart
 - MMC over SPI Driver, [402](#)
- mmcStartSequentialRead
 - MMC over SPI Driver, [405](#)
- mmcStartSequentialWrite
 - MMC over SPI Driver, [407](#)
- mmcStop
 - MMC over SPI Driver, [403](#)
- mmcStopSequentialRead
 - MMC over SPI Driver, [407](#)
- mmcStopSequentialWrite
 - MMC over SPI Driver, [409](#)
- mmcSync
 - MMC over SPI Driver, [410](#)
- mmcscdGetCardCapacity
 - MMC/SD Block Device, [419](#)
- mode
 - EXTChannelConfig, [746](#)
 - ICUConfig, [761](#)
 - PWMChannelConfig, [791](#)
- mode_detect
 - SDC Driver, [524](#)
- month
 - RTCDatetime, [799](#)
- msg_t
 - OSAL, [352](#)
- mutex
 - ADCDriver, [695](#)
 - DACDriver, [744](#)
 - QSPIDriver, [797](#)
 - SPIDriver, [819](#)
 - UARTDriver, [823](#)
- mutex_t
 - OSAL, [353](#)
- NWAYTEST_LOOPBACK
 - MII/RMII Header, [392](#)
- NWAYTEST_RESV1
 - MII/RMII Header, [392](#)
- NWAYTEST_RESV2
 - MII/RMII Header, [392](#)
- next_offset
 - MFSDriver, [780](#)
- Normal Drivers, [335](#)
- notify
 - io_buffers_queue, [765](#)
- num_channels
 - ADCConversionGroup, [693](#)
 - DACConversionGroup, [742](#)
- OSAL_DBG_ENABLE_ASSERTS
 - OSAL, [346](#)

- OSAL_DBG_ENABLE_CHECKS
 - OSAL, [346](#)
- OSAL_IRQ_EPILOGUE
 - OSAL, [348](#)
- OSAL_IRQ_HANDLER
 - OSAL, [348](#)
- OSAL_IRQ_IS_VALID_PRIORITY
 - OSAL, [348](#)
- OSAL_IRQ_MAXIMUM_PRIORITY
 - OSAL, [346](#)
- OSAL_IRQ_PRIORITY_LEVELS
 - OSAL, [346](#)
- OSAL_IRQ_PROLOGUE
 - OSAL, [348](#)
- OSAL_MS2RTC
 - OSAL, [350](#)
- OSAL_MS2I
 - OSAL, [349](#)
- OSAL_S2RTC
 - OSAL, [350](#)
- OSAL_S2I
 - OSAL, [348](#)
- OSAL_ST_FREQUENCY
 - OSAL, [346](#)
- OSAL_ST_MODE
 - OSAL, [346](#)
- OSAL_ST_RESOLUTION
 - OSAL, [346](#)
- OSAL_US2RTC
 - OSAL, [351](#)
- OSAL_US2I
 - OSAL, [349](#)
- OSAL, [341](#)
 - event_source_t, [353](#)
 - eventcallback_t, [353](#)
 - eventflags_t, [353](#)
 - msg_t, [352](#)
 - mutex_t, [353](#)
 - OSAL_DBG_ENABLE_ASSERTS, [346](#)
 - OSAL_DBG_ENABLE_CHECKS, [346](#)
 - OSAL_IRQ_EPILOGUE, [348](#)
 - OSAL_IRQ_HANDLER, [348](#)
 - OSAL_IRQ_IS_VALID_PRIORITY, [348](#)
 - OSAL_IRQ_MAXIMUM_PRIORITY, [346](#)
 - OSAL_IRQ_PRIORITY_LEVELS, [346](#)
 - OSAL_IRQ_PROLOGUE, [348](#)
 - OSAL_MS2RTC, [350](#)
 - OSAL_MS2I, [349](#)
 - OSAL_S2RTC, [350](#)
 - OSAL_S2I, [348](#)
 - OSAL_ST_FREQUENCY, [346](#)
 - OSAL_ST_MODE, [346](#)
 - OSAL_ST_RESOLUTION, [346](#)
 - OSAL_US2RTC, [351](#)
 - OSAL_US2I, [349](#)
 - osal_halt_msg, [364](#)
 - osalDbgAssert, [346](#)
 - osalDbgCheck, [347](#)
 - osalDbgCheckClassI, [348](#)
 - osalDbgCheckClassS, [348](#)
 - osalEventBroadcastFlags, [359](#)
 - osalEventBroadcastFlagsI, [358](#)
 - osalEventObjectInit, [364](#)
 - osalEventSetCallback, [359](#)
 - osalInit, [353](#)
 - osalMutexLock, [360](#)
 - osalMutexObjectInit, [364](#)
 - osalMutexUnlock, [360](#)
 - osalOsGetSystemTimeX, [354](#)
 - osalOsRescheduleS, [354](#)
 - osalOsTimerHandlerI, [354](#)
 - osalSysDisable, [360](#)
 - osalSysEnable, [360](#)
 - osalSysGetStatusAndLockX, [361](#)
 - osalSysHalt, [353](#)
 - osalSysLock, [361](#)
 - osalSysLockFromISR, [361](#)
 - osalSysPolledDelayX, [354](#)
 - osalSysRestoreStatusX, [362](#)
 - osalSysUnlock, [361](#)
 - osalSysUnlockFromISR, [361](#)
 - osalThreadDequeueAllI, [358](#)
 - osalThreadDequeueNextI, [358](#)
 - osalThreadEnqueueTimeoutS, [357](#)
 - osalThreadQueueObjectInit, [363](#)
 - osalThreadResumeI, [356](#)
 - osalThreadResumeS, [357](#)
 - osalThreadSleep, [355](#)
 - osalThreadSleepMicroseconds, [352](#)
 - osalThreadSleepMilliseconds, [351](#)
 - osalThreadSleepSeconds, [351](#)
 - osalThreadSleepS, [355](#)
 - osalThreadSuspendTimeoutS, [356](#)
 - osalThreadSuspendS, [355](#)
 - osalTimeAddX, [362](#)
 - osalTimeDiffX, [362](#)
 - osalTimeIsInRangeX, [363](#)
 - rtcnt_t, [352](#)
 - sysinterval_t, [352](#)
 - syssts_t, [352](#)
 - system_time_t, [352](#)
 - thread_reference_t, [352](#)
- objGetInstance
 - Base Object, [232](#)
- obnotify
 - Serial over USB Driver, [568](#)
- obqFlush
 - I/O Buffers Queues, [208](#)
- obqGetEmptyBufferTimeout
 - I/O Buffers Queues, [203](#)
- obqGetEmptyBufferTimeoutS
 - I/O Buffers Queues, [204](#)
- obqGetFullBufferI
 - I/O Buffers Queues, [202](#)
- obqIsEmptyI
 - I/O Buffers Queues, [192](#)

- obqIsFullI
 - I/O Buffers Queues, [193](#)
- obqObjectInit
 - I/O Buffers Queues, [200](#)
- obqPostFullBuffer
 - I/O Buffers Queues, [205](#)
- obqPostFullBufferS
 - I/O Buffers Queues, [205](#)
- obqPutTimeout
 - I/O Buffers Queues, [206](#)
- obqReleaseEmptyBufferI
 - I/O Buffers Queues, [202](#)
- obqResetI
 - I/O Buffers Queues, [201](#)
- obqTryFlushI
 - I/O Buffers Queues, [208](#)
- obqWriteTimeout
 - I/O Buffers Queues, [207](#)
- offset
 - flash_sector_descriptor_t, [754](#)
 - IOBus, [768](#)
 - MACReceiveDescriptor, [773](#)
 - MACTransmitDescriptor, [773](#)
- oq_write
 - I/O Bytes Queues, [245](#)
- oqGetEmptyI
 - I/O Bytes Queues, [242](#)
- oqGetFullI
 - I/O Bytes Queues, [242](#)
- oqGetI
 - I/O Bytes Queues, [253](#)
- oqlsEmptyI
 - I/O Bytes Queues, [242](#)
- oqlsFullI
 - I/O Bytes Queues, [243](#)
- oqObjectInit
 - I/O Bytes Queues, [250](#)
- oqPut
 - I/O Bytes Queues, [243](#)
- oqPutTimeout
 - I/O Bytes Queues, [252](#)
- oqPutI
 - I/O Bytes Queues, [251](#)
- oqResetI
 - I/O Bytes Queues, [251](#)
- oqWriteTimeout
 - I/O Bytes Queues, [254](#)
- oqWritel
 - I/O Bytes Queues, [253](#)
- osal.c, [971](#)
- osal.h, [973](#)
- osal_halt_msg
 - OSAL, [364](#)
- osalDbgAssert
 - OSAL, [346](#)
- osalDbgCheck
 - OSAL, [347](#)
- osalDbgCheckClassI
 - OSAL, [348](#)
- osalDbgCheckClassS
 - OSAL, [348](#)
- osalEventBroadcastFlags
 - OSAL, [359](#)
- osalEventBroadcastFlagsI
 - OSAL, [358](#)
- osalEventObjectInit
 - OSAL, [364](#)
- osalEventSetCallback
 - OSAL, [359](#)
- osalInit
 - OSAL, [353](#)
- osalMutexLock
 - OSAL, [360](#)
- osalMutexObjectInit
 - OSAL, [364](#)
- osalMutexUnlock
 - OSAL, [360](#)
- osalOsGetSystemTimeX
 - OSAL, [354](#)
- osalOsRescheduleS
 - OSAL, [354](#)
- osalOsTimerHandlerI
 - OSAL, [354](#)
- osalSysDisable
 - OSAL, [360](#)
- osalSysEnable
 - OSAL, [360](#)
- osalSysGetStatusAndLockX
 - OSAL, [361](#)
- osalSysHalt
 - OSAL, [353](#)
- osalSysLock
 - OSAL, [361](#)
- osalSysLockFromISR
 - OSAL, [361](#)
- osalSysPolledDelayX
 - OSAL, [354](#)
- osalSysRestoreStatusX
 - OSAL, [362](#)
- osalSysUnlock
 - OSAL, [361](#)
- osalSysUnlockFromISR
 - OSAL, [361](#)
- osalThreadDequeueAllI
 - OSAL, [358](#)
- osalThreadDequeueNextI
 - OSAL, [358](#)
- osalThreadEnqueueTimeoutS
 - OSAL, [357](#)
- osalThreadQueueObjectInit
 - OSAL, [363](#)
- osalThreadResumeI
 - OSAL, [356](#)
- osalThreadResumeS
 - OSAL, [357](#)
- osalThreadSleep

- OSAL, [355](#)
- osalThreadSleepMicroseconds
 - OSAL, [352](#)
- osalThreadSleepMilliseconds
 - OSAL, [351](#)
- osalThreadSleepSeconds
 - OSAL, [351](#)
- osalThreadSleepS
 - OSAL, [355](#)
- osalThreadSuspendTimeoutS
 - OSAL, [356](#)
- osalThreadSuspendS
 - OSAL, [355](#)
- osalTimeAddX
 - OSAL, [362](#)
- osalTimeDiffX
 - OSAL, [362](#)
- osalTimeIsInRangeX
 - OSAL, [363](#)
- out
 - hal_usb_llc, [962](#)
 - USB Driver, [682](#)
- out_cb
 - USBEndpointConfig, [835](#)
- out_maxsize
 - USBEndpointConfig, [836](#)
- out_params
 - USBDriver, [832](#)
- out_state
 - USBEndpointConfig, [836](#)
- output_buffers_queue_t
 - I/O Buffers Queues, [193](#)
- output_queue_t
 - I/O Bytes Queues, [244](#)
- overflow_cb
 - ICUConfig, [762](#)
- PAL Driver, [425](#)
 - _IOBUS_DATA, [433](#)
 - _pal_clear_event, [434](#), [435](#)
 - _pal_init_event, [433](#), [434](#)
 - _pal_isr_code, [436](#)
 - _pal_llc_init, [463](#)
 - _pal_llc_setgroupmode, [463](#)
 - IOBUS_DECL, [433](#)
 - IOPORT1, [450](#)
 - ioline_t, [458](#)
 - iomode_t, [458](#)
 - iopadid_t, [458](#)
 - ioportid_t, [458](#)
 - ioportmask_t, [458](#)
 - PAL_EVENT_MODE_BOTH_EDGES, [432](#)
 - PAL_EVENT_MODE_DISABLED, [432](#)
 - PAL_EVENT_MODE_EDGES_MASK, [432](#)
 - PAL_EVENT_MODE_FALLING_EDGE, [432](#)
 - PAL_EVENT_MODE_RISING_EDGE, [432](#)
 - PAL_GROUP_MASK, [433](#)
 - PAL_HIGH, [432](#)
 - PAL_IOPORTS_WIDTH, [450](#)
 - PAL_LINE, [450](#)
 - PAL_LOW, [431](#)
 - PAL_MODE_INPUT_ANALOG, [431](#)
 - PAL_MODE_INPUT_PULLDOWN, [431](#)
 - PAL_MODE_INPUT_PULLUP, [431](#)
 - PAL_MODE_INPUT, [431](#)
 - PAL_MODE_OUTPUT_OPENDRAIN, [431](#)
 - PAL_MODE_OUTPUT_PUSHPULL, [431](#)
 - PAL_MODE_RESET, [431](#)
 - PAL_MODE_UNCONNECTED, [431](#)
 - PAL_NOLINE, [450](#)
 - PAL_PAD, [450](#)
 - PAL_PORT_BIT, [432](#)
 - PAL_PORT, [450](#)
 - PAL_USE_CALLBACKS, [432](#)
 - PAL_USE_WAIT, [432](#)
 - PAL_WHOLE_PORT, [450](#)
 - pal_llc_clearpad, [456](#)
 - pal_llc_clearport, [452](#)
 - pal_llc_get_line_event, [457](#)
 - pal_llc_get_pad_event, [457](#)
 - pal_llc_init, [450](#)
 - pal_llc_readgroup, [453](#)
 - pal_llc_readlatch, [451](#)
 - pal_llc_readpad, [454](#)
 - pal_llc_readport, [450](#)
 - pal_llc_setgroupmode, [454](#)
 - pal_llc_setpad, [455](#)
 - pal_llc_setpadmode, [457](#)
 - pal_llc_setport, [451](#)
 - pal_llc_togglepad, [456](#)
 - pal_llc_toggleport, [452](#)
 - pal_llc_writegroup, [453](#)
 - pal_llc_writepad, [455](#)
 - pal_llc_writeport, [451](#)
 - palClearLine, [444](#)
 - palClearPad, [442](#)
 - palClearPort, [439](#)
 - palDisableLineEvent, [448](#)
 - palDisableLineEventI, [447](#)
 - palDisablePadEvent, [447](#)
 - palDisablePadEventI, [446](#)
 - palEnableLineEvent, [448](#)
 - palEnableLineEventI, [447](#)
 - palEnablePadEvent, [446](#)
 - palEnablePadEventI, [446](#)
 - palInit, [437](#)
 - palReadBus, [458](#)
 - palReadGroup, [439](#)
 - palReadLatch, [437](#)
 - palReadLine, [443](#)
 - palReadPad, [441](#)
 - palReadPort, [437](#)
 - palSetBusMode, [459](#)
 - palSetGroupMode, [440](#)
 - palSetLine, [444](#)
 - palSetLineCallback, [449](#)
 - palSetLineCallbackI, [460](#)

- palSetLineMode, [445](#)
- palSetPad, [442](#)
- palSetPadCallback, [449](#)
- palSetPadCallbackI, [459](#)
- palSetPadMode, [443](#)
- palSetPort, [438](#)
- palToggleLine, [445](#)
- palTogglePad, [442](#)
- palTogglePort, [439](#)
- palWaitLineTimeout, [462](#)
- palWaitLineTimeoutS, [462](#)
- palWaitPadTimeout, [461](#)
- palWaitPadTimeoutS, [460](#)
- palWriteBus, [459](#)
- palWriteGroup, [440](#)
- palWriteLine, [444](#)
- palWritePad, [441](#)
- palWritePort, [438](#)
- palcallback_t, [458](#)
- PAL_EVENT_MODE_BOTH_EDGES
 - PAL Driver, [432](#)
- PAL_EVENT_MODE_DISABLED
 - PAL Driver, [432](#)
- PAL_EVENT_MODE_EDGES_MASK
 - PAL Driver, [432](#)
- PAL_EVENT_MODE_FALLING_EDGE
 - PAL Driver, [432](#)
- PAL_EVENT_MODE_RISING_EDGE
 - PAL Driver, [432](#)
- PAL_GROUP_MASK
 - PAL Driver, [433](#)
- PAL_HIGH
 - PAL Driver, [432](#)
- PAL_IOPORTS_WIDTH
 - PAL Driver, [450](#)
- PAL_LINE
 - PAL Driver, [450](#)
- PAL_LOW
 - PAL Driver, [431](#)
- PAL_MODE_INPUT_ANALOG
 - PAL Driver, [431](#)
- PAL_MODE_INPUT_PULLDOWN
 - PAL Driver, [431](#)
- PAL_MODE_INPUT_PULLUP
 - PAL Driver, [431](#)
- PAL_MODE_INPUT
 - PAL Driver, [431](#)
- PAL_MODE_OUTPUT_OPENDRAIN
 - PAL Driver, [431](#)
- PAL_MODE_OUTPUT_PUSHPULL
 - PAL Driver, [431](#)
- PAL_MODE_RESET
 - PAL Driver, [431](#)
- PAL_MODE_UNCONNECTED
 - PAL Driver, [431](#)
- PAL_NOLINE
 - PAL Driver, [450](#)
- PAL_PAD
 - PAL Driver, [450](#)
- PAL_PORT_BIT
 - PAL Driver, [432](#)
- PAL_PORT
 - PAL Driver, [450](#)
- PAL_USE_CALLBACKS
 - Configuration, [330](#)
 - PAL Driver, [432](#)
- PAL_USE_WAIT
 - Configuration, [330](#)
 - PAL Driver, [432](#)
- PAL_WHOLE_PORT
 - PAL Driver, [450](#)
- PALConfig, [789](#)
- PLATFORM_ADC_USE_ADC1
 - ADC Driver, [34](#)
- PLATFORM_CAN_USE_CAN1
 - CAN Driver, [55](#)
- PLATFORM_CRY_USE_CRY1
 - Crypto Driver, [78](#)
- PLATFORM_DAC_USE_DAC1
 - DAC Driver, [133](#)
- PLATFORM_EXT_USE_EXT1
 - EXT Driver, [149](#)
- PLATFORM_GPT_USE_GPT1
 - GPT Driver, [168](#)
- PLATFORM_I2C_USE_I2C1
 - I2C Driver, [268](#)
- PLATFORM_I2S_USE_I2S1
 - I2S Driver, [281](#)
- PLATFORM_ICU_USE_ICU1
 - ICU Driver, [294](#)
- PLATFORM_MAC_USE_MAC1
 - MAC Driver, [311](#)
- PLATFORM_PWM_USE_PWM1
 - PWM Driver, [475](#)
- PLATFORM_QSPI_USE_QSPI1
 - QSPI Driver, [495](#)
- PLATFORM_RTC_USE_RTC1
 - RTC Driver, [510](#)
- PLATFORM_SDC_USE_SDC1
 - SDC Driver, [523](#)
- PLATFORM_SERIAL_USE_USART1
 - Serial Driver, [556](#)
- PLATFORM_SPI_USE_SPI1
 - SPI Driver, [586](#)
- PLATFORM_UART_USE_UART1
 - UART Driver, [626](#)
- PLATFORM_USB_USE_USB1
 - USB Driver, [658](#)
- PLATFORM_WDG_USE_WDG1
 - WDG Driver, [686](#)
- PWM Driver, [466](#)
 - PLATFORM_PWM_USE_PWM1, [475](#)
 - PWM_CHANNELS, [475](#)
 - PWM_DEGREES_TO_WIDTH, [470](#)
 - PWM_FRACTION_TO_WIDTH, [469](#)
 - PWM_OUTPUT_ACTIVE_HIGH, [469](#)

- PWM_OUTPUT_ACTIVE_LOW, [469](#)
- PWM_OUTPUT_DISABLED, [469](#)
- PWM_OUTPUT_MASK, [469](#)
- PWM_PERCENTAGE_TO_WIDTH, [470](#)
- PWM_READY, [477](#)
- PWM_STOP, [477](#)
- PWM_UNINIT, [477](#)
- PWMD1, [488](#)
- PWMDriver, [476](#)
- pwm_ild_change_period, [475](#)
- pwm_ild_disable_channel, [486](#)
- pwm_ild_disable_channel_notification, [487](#)
- pwm_ild_disable_periodic_notification, [486](#)
- pwm_ild_enable_channel, [485](#)
- pwm_ild_enable_channel_notification, [487](#)
- pwm_ild_enable_periodic_notification, [486](#)
- pwm_ild_init, [484](#)
- pwm_ild_start, [484](#)
- pwm_ild_stop, [485](#)
- pwmChangePeriod, [479](#)
- pwmChangePeriodI, [471](#)
- pwmDisableChannel, [481](#)
- pwmDisableChannelNotification, [483](#)
- pwmDisableChannelNotificationI, [475](#)
- pwmDisableChannell, [472](#)
- pwmDisablePeriodicNotification, [482](#)
- pwmDisablePeriodicNotificationI, [474](#)
- pwmEnableChannel, [480](#)
- pwmEnableChannelNotification, [483](#)
- pwmEnableChannelNotificationI, [474](#)
- pwmEnableChannell, [472](#)
- pwmEnablePeriodicNotification, [482](#)
- pwmEnablePeriodicNotificationI, [473](#)
- pwmInit, [477](#)
- pwmIsChannelEnabledI, [473](#)
- pwmObjectInit, [477](#)
- pwmStart, [477](#)
- pwmStop, [478](#)
- pwmcallback_t, [476](#)
- pwmchannel_t, [476](#)
- pwmchnmsk_t, [476](#)
- pwmcnt_t, [476](#)
- pwmmode_t, [476](#)
- pwmstate_t, [477](#)
- PWM_CHANNELS
 - PWM Driver, [475](#)
- PWM_DEGREES_TO_WIDTH
 - PWM Driver, [470](#)
- PWM_FRACTION_TO_WIDTH
 - PWM Driver, [469](#)
- PWM_OUTPUT_ACTIVE_HIGH
 - PWM Driver, [469](#)
- PWM_OUTPUT_ACTIVE_LOW
 - PWM Driver, [469](#)
- PWM_OUTPUT_DISABLED
 - PWM Driver, [469](#)
- PWM_OUTPUT_MASK
 - PWM Driver, [469](#)
- PWM_PERCENTAGE_TO_WIDTH
 - PWM Driver, [470](#)
- PWM_READY
 - PWM Driver, [477](#)
- PWM_STOP
 - PWM Driver, [477](#)
- PWM_UNINIT
 - PWM Driver, [477](#)
- PWMChannelConfig, [791](#)
 - callback, [791](#)
 - mode, [791](#)
- PWMConfig, [792](#)
 - callback, [793](#)
 - channels, [793](#)
 - frequency, [793](#)
 - period, [793](#)
- PWMD1
 - PWM Driver, [488](#)
- PWMDriver, [793](#)
 - channels, [795](#)
 - config, [795](#)
 - enabled, [795](#)
 - PWM Driver, [476](#)
 - period, [795](#)
 - state, [794](#)
- page_size
 - flash_descriptor_t, [752](#)
- pal_ild_clearpad
 - PAL Driver, [456](#)
- pal_ild_clearport
 - PAL Driver, [452](#)
- pal_ild_get_line_event
 - PAL Driver, [457](#)
- pal_ild_get_pad_event
 - PAL Driver, [457](#)
- pal_ild_init
 - PAL Driver, [450](#)
- pal_ild_readgroup
 - PAL Driver, [453](#)
- pal_ild_readlatch
 - PAL Driver, [451](#)
- pal_ild_readpad
 - PAL Driver, [454](#)
- pal_ild_readport
 - PAL Driver, [450](#)
- pal_ild_setgroupmode
 - PAL Driver, [454](#)
- pal_ild_setpad
 - PAL Driver, [455](#)
- pal_ild_setpadmode
 - PAL Driver, [457](#)
- pal_ild_setport
 - PAL Driver, [451](#)
- pal_ild_togglepad
 - PAL Driver, [456](#)
- pal_ild_toggleport
 - PAL Driver, [452](#)
- pal_ild_writigroup

- PAL Driver, [453](#)
- pal_lld_writepad
 - PAL Driver, [455](#)
- pal_lld_writeport
 - PAL Driver, [451](#)
- palClearLine
 - PAL Driver, [444](#)
- palClearPad
 - PAL Driver, [442](#)
- palClearPort
 - PAL Driver, [439](#)
- palDisableLineEvent
 - PAL Driver, [448](#)
- palDisableLineEventI
 - PAL Driver, [447](#)
- palDisablePadEvent
 - PAL Driver, [447](#)
- palDisablePadEventI
 - PAL Driver, [446](#)
- palEnableLineEvent
 - PAL Driver, [448](#)
- palEnableLineEventI
 - PAL Driver, [447](#)
- palEnablePadEvent
 - PAL Driver, [446](#)
- palEnablePadEventI
 - PAL Driver, [446](#)
- palInit
 - PAL Driver, [437](#)
- palReadBus
 - PAL Driver, [458](#)
- palReadGroup
 - PAL Driver, [439](#)
- palReadLatch
 - PAL Driver, [437](#)
- palReadLine
 - PAL Driver, [443](#)
- palReadPad
 - PAL Driver, [441](#)
- palReadPort
 - PAL Driver, [437](#)
- palSetBusMode
 - PAL Driver, [459](#)
- palSetGroupMode
 - PAL Driver, [440](#)
- palSetLine
 - PAL Driver, [444](#)
- palSetLineCallback
 - PAL Driver, [449](#)
- palSetLineCallbackI
 - PAL Driver, [460](#)
- palSetLineMode
 - PAL Driver, [445](#)
- palSetPad
 - PAL Driver, [442](#)
- palSetPadCallback
 - PAL Driver, [449](#)
- palSetPadCallbackI
 - PAL Driver, [459](#)
- palSetPadMode
 - PAL Driver, [443](#)
- palSetPort
 - PAL Driver, [438](#)
- palToggleLine
 - PAL Driver, [445](#)
- palTogglePad
 - PAL Driver, [442](#)
- palTogglePort
 - PAL Driver, [439](#)
- palWaitLineTimeout
 - PAL Driver, [462](#)
- palWaitLineTimeoutS
 - PAL Driver, [462](#)
- palWaitPadTimeout
 - PAL Driver, [461](#)
- palWaitPadTimeoutS
 - PAL Driver, [460](#)
- palWriteBus
 - PAL Driver, [459](#)
- palWriteGroup
 - PAL Driver, [440](#)
- palWriteLine
 - PAL Driver, [444](#)
- palWritePad
 - PAL Driver, [441](#)
- palWritePort
 - PAL Driver, [438](#)
- palcallback_t
 - PAL Driver, [458](#)
- palevent_t, [789](#)
 - arg, [790](#)
 - cb, [790](#)
 - threads, [790](#)
- param
 - event_source, [745](#)
- period
 - PWMConfig, [793](#)
 - PWMDriver, [795](#)
- period_cb
 - ICUConfig, [762](#)
- portid
 - IOBus, [768](#)
- ptr
 - io_buffers_queue, [765](#)
- pwm_lld_change_period
 - PWM Driver, [475](#)
- pwm_lld_disable_channel
 - PWM Driver, [486](#)
- pwm_lld_disable_channel_notification
 - PWM Driver, [487](#)
- pwm_lld_disable_periodic_notification
 - PWM Driver, [486](#)
- pwm_lld_enable_channel
 - PWM Driver, [485](#)
- pwm_lld_enable_channel_notification
 - PWM Driver, [487](#)

- pwm_lld_enable_periodic_notification
 - PWM Driver, [486](#)
- pwm_lld_init
 - PWM Driver, [484](#)
- pwm_lld_start
 - PWM Driver, [484](#)
- pwm_lld_stop
 - PWM Driver, [485](#)
- pwmChangePeriod
 - PWM Driver, [479](#)
- pwmChangePeriodl
 - PWM Driver, [471](#)
- pwmDisableChannel
 - PWM Driver, [481](#)
- pwmDisableChannelNotification
 - PWM Driver, [483](#)
- pwmDisableChannelNotificationl
 - PWM Driver, [475](#)
- pwmDisableChannell
 - PWM Driver, [472](#)
- pwmDisablePeriodicNotification
 - PWM Driver, [482](#)
- pwmDisablePeriodicNotificationl
 - PWM Driver, [474](#)
- pwmEnableChannel
 - PWM Driver, [480](#)
- pwmEnableChannelNotification
 - PWM Driver, [483](#)
- pwmEnableChannelNotificationl
 - PWM Driver, [474](#)
- pwmEnableChannell
 - PWM Driver, [472](#)
- pwmEnablePeriodicNotification
 - PWM Driver, [482](#)
- pwmEnablePeriodicNotificationl
 - PWM Driver, [473](#)
- pwmInit
 - PWM Driver, [477](#)
- pwmIsChannelEnabledl
 - PWM Driver, [473](#)
- pwmObjectInit
 - PWM Driver, [477](#)
- pwmStart
 - PWM Driver, [477](#)
- pwmStop
 - PWM Driver, [478](#)
- pwmcallback_t
 - PWM Driver, [476](#)
- pwmchannel_t
 - PWM Driver, [476](#)
- pwmchnmsk_t
 - PWM Driver, [476](#)
- pwmcnt_t
 - PWM Driver, [476](#)
- pwmmode_t
 - PWM Driver, [476](#)
- pwmstate_t
 - PWM Driver, [477](#)
- Q_EMPTY
 - I/O Bytes Queues, [238](#)
- Q_FULL
 - I/O Bytes Queues, [238](#)
- Q_OK
 - I/O Bytes Queues, [238](#)
- Q_RESET
 - I/O Bytes Queues, [238](#)
- Q_TIMEOUT
 - I/O Bytes Queues, [238](#)
- q_buffer
 - io_queue, [767](#)
- q_counter
 - io_queue, [767](#)
- q_link
 - io_queue, [767](#)
- q_notify
 - io_queue, [767](#)
- q_rdprr
 - io_queue, [767](#)
- q_top
 - io_queue, [767](#)
- q_waiting
 - io_queue, [767](#)
- q_wrptr
 - io_queue, [767](#)
- qGetLink
 - I/O Bytes Queues, [239](#)
- QSPI Driver, [489](#)
 - _qspi_isr_code, [495](#)
 - _qspi_wakeup_isr, [494](#)
 - PLATFORM_QSPI_USE_QSPI1, [495](#)
 - QSPI_ACTIVE, [496](#)
 - QSPI_COMPLETE, [496](#)
 - QSPI_MEMMAP, [496](#)
 - QSPI_READY, [496](#)
 - QSPI_STOP, [496](#)
 - QSPI_UNINIT, [496](#)
 - QSPI_USE_MUTUAL_EXCLUSION, [492](#)
 - QSPI_USE_WAIT, [492](#)
 - QSPID1, [507](#)
 - QSPIDriver, [495](#)
 - qspi_lld_command, [505](#)
 - qspi_lld_init, [505](#)
 - qspi_lld_map_flash, [506](#)
 - qspi_lld_receive, [506](#)
 - qspi_lld_send, [506](#)
 - qspi_lld_start, [505](#)
 - qspi_lld_stop, [505](#)
 - qspi_lld_unmap_flash, [507](#)
 - qspiAcquireBus, [503](#)
 - qspiCommand, [500](#)
 - qspiInit, [496](#)
 - qspiMapFlash, [502](#)
 - qspiMapFlashl, [493](#)
 - qspiObjectInit, [496](#)
 - qspiReceive, [502](#)
 - qspiReleaseBus, [504](#)

- qspiSend, [501](#)
- qspiStart, [497](#)
- qspiStartCommand, [498](#)
- qspiStartCommandI, [492](#)
- qspiStartReceive, [499](#)
- qspiStartReceiveI, [493](#)
- qspiStartSend, [499](#)
- qspiStartSendI, [492](#)
- qspiStop, [497](#)
- qspiUnmapFlash, [503](#)
- qspiUnmapFlashI, [494](#)
- qspicallback_t, [495](#)
- qspistate_t, [496](#)
- QSPI_ACTIVE
 - QSPI Driver, [496](#)
- QSPI_COMPLETE
 - QSPI Driver, [496](#)
- QSPI_MEMMAP
 - QSPI Driver, [496](#)
- QSPI_READY
 - QSPI Driver, [496](#)
- QSPI_STOP
 - QSPI Driver, [496](#)
- QSPI_UNINIT
 - QSPI Driver, [496](#)
- QSPI_USE_MUTUAL_EXCLUSION
 - Configuration, [332](#)
 - QSPI Driver, [492](#)
- QSPI_USE_WAIT
 - Configuration, [331](#)
 - QSPI Driver, [492](#)
- QSPIDriver, [795](#)
 - end_cb, [796](#)
- QSPID1
 - QSPI Driver, [507](#)
- QSPIDriver, [796](#)
 - config, [797](#)
 - mutex, [797](#)
 - QSPI Driver, [495](#)
 - state, [797](#)
 - thread, [797](#)
- qSizeX
 - I/O Bytes Queues, [238](#)
- qSpacel
 - I/O Bytes Queues, [238](#)
- qnotify_t
 - I/O Bytes Queues, [244](#)
- qspi_command_t, [795](#)
- qspi_llid_command
 - QSPI Driver, [505](#)
- qspi_llid_init
 - QSPI Driver, [505](#)
- qspi_llid_map_flash
 - QSPI Driver, [506](#)
- qspi_llid_receive
 - QSPI Driver, [506](#)
- qspi_llid_send
 - QSPI Driver, [506](#)
- qspi_llid_start
 - QSPI Driver, [505](#)
- qspi_llid_stop
 - QSPI Driver, [505](#)
- qspi_llid_unmap_flash
 - QSPI Driver, [507](#)
- qspiAcquireBus
 - QSPI Driver, [503](#)
- qspiCommand
 - QSPI Driver, [500](#)
- qspiInit
 - QSPI Driver, [496](#)
- qspiMapFlash
 - QSPI Driver, [502](#)
- qspiMapFlashI
 - QSPI Driver, [493](#)
- qspiObjectInit
 - QSPI Driver, [496](#)
- qspiReceive
 - QSPI Driver, [502](#)
- qspiReleaseBus
 - QSPI Driver, [504](#)
- qspiSend
 - QSPI Driver, [501](#)
- qspiStart
 - QSPI Driver, [497](#)
- qspiStartCommand
 - QSPI Driver, [498](#)
- qspiStartCommandI
 - QSPI Driver, [492](#)
- qspiStartReceive
 - QSPI Driver, [499](#)
- qspiStartReceiveI
 - QSPI Driver, [493](#)
- qspiStartSend
 - QSPI Driver, [499](#)
- qspiStartSendI
 - QSPI Driver, [492](#)
- qspiStop
 - QSPI Driver, [497](#)
- qspiUnmapFlash
 - QSPI Driver, [503](#)
- qspiUnmapFlashI
 - QSPI Driver, [494](#)
- qspicallback_t
 - QSPI Driver, [495](#)
- qspistate_t
 - QSPI Driver, [496](#)
- RET_ON_ERROR
 - Managed Flash Storage Driver, [367](#)
- RTC Driver, [508](#)
 - _rtc_driver_methods, [510](#)
 - PLATFORM_RTC_USE_RTC1, [510](#)
 - RTC_ALARMS, [510](#)
 - RTC_BASE_YEAR, [510](#)
 - RTC_HAS_STORAGE, [510](#)
 - RTC_SUPPORTS_CALLBACKS, [510](#)
 - RTCDriver, [510](#)

- rtc_lld_get_alarm, [517](#)
- rtc_lld_get_time, [516](#)
- rtc_lld_init, [515](#)
- rtc_lld_set_alarm, [516](#)
- rtc_lld_set_time, [516](#)
- rtcConvertDateTimeToFAT, [515](#)
- rtcConvertDateTimeToStructTm, [514](#)
- rtcConvertStructTmToDateTime, [515](#)
- rtcGetAlarm, [513](#)
- rtcGetTime, [512](#)
- rtclnit, [511](#)
- rtcObjectInit, [511](#)
- rtcSetAlarm, [513](#)
- rtcSetCallback, [514](#)
- rtcSetTime, [511](#)
- rtcalarm_t, [510](#)
- rtccb_t, [510](#)
- rtcevent_t, [511](#)
- RTC_ALARMS
 - RTC Driver, [510](#)
- RTC_BASE_YEAR
 - RTC Driver, [510](#)
- RTC_HAS_STORAGE
 - RTC Driver, [510](#)
- RTC_SUPPORTS_CALLBACKS
 - RTC Driver, [510](#)
- RTCAlarm, [798](#)
- RTCDateTime, [798](#)
 - day, [799](#)
 - dayofweek, [799](#)
 - dstflag, [799](#)
 - millisecond, [799](#)
 - month, [799](#)
 - year, [799](#)
- RTCDriver, [800](#)
 - RTC Driver, [510](#)
 - vmt, [800](#)
- RTCDriverVMT, [800](#)
- RTR
 - CANRxFrame, [736](#)
 - CANTxFrame, [738](#)
- rca
 - SDCDriver, [805](#)
- rdevent
 - MACDriver, [772](#)
- rdqueue
 - MACDriver, [772](#)
- read_CxD
 - MMC over SPI Driver, [400](#)
- receiving
 - USBDriver, [831](#)
- recvr1
 - MMC over SPI Driver, [397](#)
- recvr3
 - MMC over SPI Driver, [398](#)
- requests_hook_cb
 - USBConfig, [829](#)
- reserved1
 - mfs_bank_header_t, [774](#)
- rtc_lld_get_alarm
 - RTC Driver, [517](#)
- rtc_lld_get_time
 - RTC Driver, [516](#)
- rtc_lld_init
 - RTC Driver, [515](#)
- rtc_lld_set_alarm
 - RTC Driver, [516](#)
- rtc_lld_set_time
 - RTC Driver, [516](#)
- rtcConvertDateTimeToFAT
 - RTC Driver, [515](#)
- rtcConvertDateTimeToStructTm
 - RTC Driver, [514](#)
- rtcConvertStructTmToDateTime
 - RTC Driver, [515](#)
- rtcGetAlarm
 - RTC Driver, [513](#)
- rtcGetTime
 - RTC Driver, [512](#)
- rtclnit
 - RTC Driver, [511](#)
- rtcObjectInit
 - RTC Driver, [511](#)
- rtcSetAlarm
 - RTC Driver, [513](#)
- rtcSetCallback
 - RTC Driver, [514](#)
- rtcSetTime
 - RTC Driver, [511](#)
- rtcalarm_t
 - RTC Driver, [510](#)
- rtccb_t
 - RTC Driver, [510](#)
- rtcevent_t
 - RTC Driver, [511](#)
- rtcnt_t
 - OSAL, [352](#)
- rx_buffer
 - I2SConfig, [759](#)
- rxbuf
 - USBOutEndpointState, [838](#)
- rxchar_cb
 - UARTConfig, [821](#)
- rxcnt
 - USBOutEndpointState, [838](#)
- rxend_cb
 - UARTConfig, [821](#)
- rxerr_cb
 - UARTConfig, [821](#)
- rxfull_event
 - CANDriver, [734](#)
- rxqueue
 - CANDriver, [734](#)
- rxsize
 - USBOutEndpointState, [838](#)
- rxstate

- UARTDriver, [823](#)
- SD1
 - Serial Driver, [564](#)
- SD_BREAK_DETECTED
 - Serial Driver, [552](#)
- SD_FRAMING_ERROR
 - Serial Driver, [552](#)
- SD_NOISE_ERROR
 - Serial Driver, [552](#)
- SD_OVERRUN_ERROR
 - Serial Driver, [552](#)
- SD_PARITY_ERROR
 - Serial Driver, [552](#)
- SD_QUEUE_FULL_ERROR
 - Serial Driver, [552](#)
- SD_READY
 - Serial Driver, [557](#)
- SD_STOP
 - Serial Driver, [557](#)
- SD_UNINIT
 - Serial Driver, [557](#)
- SDC Driver, [518](#)
 - _sdc_driver_methods, [523](#)
 - _sdc_wait_for_transfer_state, [531](#)
 - detect_bus_clk, [529](#)
 - mmc_cmd6_construct, [526](#)
 - mmc_detect_bus_clk, [528](#)
 - mmc_init, [525](#)
 - mmc_set_bus_width, [531](#)
 - mmc_switch_t, [523](#)
 - mode_detect, [524](#)
 - PLATFORM_SDC_USE_SDC1, [523](#)
 - SDC_INIT_OCR_V20, [522](#)
 - SDC_INIT_OCR, [522](#)
 - SDC_INIT_RETRY, [521](#)
 - SDC_MMC_SUPPORT, [521](#)
 - SDC_NICE_WAITING, [521](#)
 - SDCD1, [545](#)
 - SDCDriver, [523](#)
 - sd_switch_function_t, [524](#)
 - sd_switch_t, [523](#)
 - sdc_cmd6_check_status, [527](#)
 - sdc_cmd6_construct, [526](#)
 - sdc_cmd6_extract_info, [527](#)
 - sdc_detect_bus_clk, [527](#)
 - sdc_init, [525](#)
 - sdc_llid_init, [540](#)
 - sdc_llid_read, [544](#)
 - sdc_llid_send_cmd_long_crc, [543](#)
 - sdc_llid_send_cmd_none, [542](#)
 - sdc_llid_send_cmd_short, [542](#)
 - sdc_llid_send_cmd_short_crc, [543](#)
 - sdc_llid_set_bus_mode, [541](#)
 - sdc_llid_set_data_clk, [541](#)
 - sdc_llid_start, [540](#)
 - sdc_llid_start_clk, [541](#)
 - sdc_llid_stop, [540](#)
 - sdc_llid_stop_clk, [541](#)
 - sdc_llid_sync, [545](#)
 - sdc_llid_write, [544](#)
 - sdc_set_bus_width, [530](#)
 - sdc_vmt, [545](#)
 - sdcConnect, [534](#)
 - sdcDisconnect, [535](#)
 - sdcErase, [539](#)
 - sdcGetAndClearErrors, [537](#)
 - sdcGetInfo, [539](#)
 - sdcInit, [532](#)
 - sdcIsCardInserted, [522](#)
 - sdcIsWriteProtected, [522](#)
 - sdcObjectInit, [533](#)
 - sdcRead, [536](#)
 - sdcStart, [533](#)
 - sdcStop, [533](#)
 - sdcSync, [538](#)
 - sdcWrite, [537](#)
 - sdcbusclk_t, [524](#)
 - sdcbusmode_t, [524](#)
 - sdcflags_t, [523](#)
 - sdcmode_t, [523](#)
- SDC_INIT_OCR_V20
 - Configuration, [332](#)
 - SDC Driver, [522](#)
- SDC_INIT_OCR
 - Configuration, [332](#)
 - SDC Driver, [522](#)
- SDC_INIT_RETRY
 - Configuration, [332](#)
 - SDC Driver, [521](#)
- SDC_MMC_SUPPORT
 - Configuration, [332](#)
 - SDC Driver, [521](#)
- SDC_NICE_WAITING
 - Configuration, [332](#)
 - SDC Driver, [521](#)
- SDCConfig, [802](#)
 - bus_width, [803](#)
 - scratchpad, [803](#)
- SDCD1
 - SDC Driver, [545](#)
- SDCDriver, [803](#)
 - cardmode, [805](#)
 - config, [804](#)
 - errors, [805](#)
 - rca, [805](#)
 - SDC Driver, [523](#)
 - vmt, [804](#)
- SDCDriverVMT, [805](#)
- SDU_READY
 - Serial over USB Driver, [568](#)
- SDU_STOP
 - Serial over USB Driver, [568](#)
- SDU_UNINIT
 - Serial over USB Driver, [568](#)
- SERIAL_BUFFERS_SIZE
 - Configuration, [332](#)

- Serial Driver, [552](#)
- SERIAL_DEFAULT_BITRATE
 - Configuration, [332](#)
 - Serial Driver, [552](#)
- SERIAL_USB_BUFFERS_NUMBER
 - Configuration, [333](#)
 - Serial over USB Driver, [567](#)
- SERIAL_USB_BUFFERS_SIZE
 - Configuration, [333](#)
 - Serial over USB Driver, [567](#)
- SHA1Context, [815](#)
- SHA256Context, [815](#)
- SHA512Context, [816](#)
- SID
 - CANRxFrame, [736](#)
 - CANTxFrame, [738](#)
- SPI Driver, [576](#)
 - _spi_isr_code, [584](#)
 - _spi_isr_code_half1, [584](#)
 - _spi_isr_code_half2, [585](#)
 - _spi_wakeup_isr, [583](#)
 - PLATFORM_SPI_USE_SPI1, [586](#)
 - SPI_ACTIVE, [586](#)
 - SPI_COMPLETE, [586](#)
 - SPI_READY, [586](#)
 - SPI_SELECT_MODE, [580](#)
 - SPI_STOP, [586](#)
 - SPI_SUPPORTS_CIRCULAR, [586](#)
 - SPI_UNINIT, [586](#)
 - SPI_USE_CIRCULAR, [579](#)
 - SPI_USE_MUTUAL_EXCLUSION, [579](#)
 - SPI_USE_WAIT, [579](#)
 - SPID1, [603](#)
 - SPIDriver, [586](#)
 - spi_llc_abort, [602](#)
 - spi_llc_exchange, [601](#)
 - spi_llc_ignore, [600](#)
 - spi_llc_init, [599](#)
 - spi_llc_polled_exchange, [602](#)
 - spi_llc_receive, [602](#)
 - spi_llc_select, [600](#)
 - spi_llc_send, [601](#)
 - spi_llc_start, [599](#)
 - spi_llc_stop, [600](#)
 - spi_llc_unselect, [600](#)
 - spiAbort, [593](#)
 - spiAbortI, [593](#)
 - spiAcquireBus, [598](#)
 - spiExchange, [595](#)
 - spiIgnore, [594](#)
 - spiInit, [586](#)
 - spiObjectInit, [587](#)
 - spiPolledExchange, [583](#)
 - spiReceive, [597](#)
 - spiReleaseBus, [598](#)
 - spiSelect, [588](#)
 - spiSelectI, [580](#)
 - spiSend, [596](#)
 - spiStart, [587](#)
 - spiStartExchange, [590](#)
 - spiStartExchangeI, [581](#)
 - spiStartIgnore, [589](#)
 - spiStartIgnoreI, [581](#)
 - spiStartReceive, [592](#)
 - spiStartReceiveI, [582](#)
 - spiStartSend, [591](#)
 - spiStartSendI, [582](#)
 - spiStop, [588](#)
 - spiUnselect, [589](#)
 - spiUnselectI, [580](#)
 - spicallback_t, [586](#)
 - spistate_t, [586](#)
- SPI_ACTIVE
 - SPI Driver, [586](#)
- SPI_COMPLETE
 - SPI Driver, [586](#)
- SPI_READY
 - SPI Driver, [586](#)
- SPI_SELECT_MODE
 - Configuration, [333](#)
 - SPI Driver, [580](#)
- SPI_STOP
 - SPI Driver, [586](#)
- SPI_SUPPORTS_CIRCULAR
 - SPI Driver, [586](#)
- SPI_UNINIT
 - SPI Driver, [586](#)
- SPI_USE_CIRCULAR
 - Configuration, [333](#)
 - SPI Driver, [579](#)
- SPI_USE_MUTUAL_EXCLUSION
 - Configuration, [333](#)
 - SPI Driver, [579](#)
- SPI_USE_WAIT
 - Configuration, [333](#)
 - SPI Driver, [579](#)
- SPIConfig, [816](#)
 - circular, [817](#)
 - end_cb, [817](#)
 - ssline, [817](#)
 - ssmask, [817](#)
 - sspad, [817](#)
 - ssport, [817](#)
- SPID1
 - SPI Driver, [603](#)
- SPIDriver, [818](#)
 - config, [819](#)
 - mutex, [819](#)
 - SPI Driver, [586](#)
 - state, [818](#)
 - thread, [819](#)
- ST Driver, [604](#)
 - st_llc_get_alarm, [609](#)
 - st_llc_get_counter, [608](#)
 - st_llc_init, [607](#)
 - st_llc_is_alarm_active, [609](#)

- st_lld_set_alarm, 608
- st_lld_start_alarm, 608
- st_lld_stop_alarm, 608
- stGetAlarm, 607
- stGetCounter, 604
- stInit, 605
- stIsAlarmActive, 605
- stSetAlarm, 606
- stStartAlarm, 605
- stStopAlarm, 606
- samples
 - ADCDriver, 695
 - DACDriver, 744
- saved_state
 - USBDriver, 832
- scratchpad
 - SDCConfig, 803
- sd_lld_init
 - Serial Driver, 563
- sd_lld_start
 - Serial Driver, 563
- sd_lld_stop
 - Serial Driver, 563
- sd_switch_function_t
 - SDC Driver, 524
- sd_switch_t
 - SDC Driver, 523
- sdAsynchronousRead
 - Serial Driver, 556
- sdAsynchronousWrite
 - Serial Driver, 555
- sdControl
 - Serial Driver, 562
- sdGet
 - Serial Driver, 553
- sdGetTimeout
 - Serial Driver, 554
- sdGetWouldBlock
 - Serial Driver, 561
- sdGetl
 - Serial Driver, 553
- sdIncomingDataI
 - Serial Driver, 559
- sdInit
 - Serial Driver, 557
- sdObjectInit
 - Serial Driver, 557
- sdPut
 - Serial Driver, 553
- sdPutTimeout
 - Serial Driver, 553
- sdPutWouldBlock
 - Serial Driver, 560
- sdPutl
 - Serial Driver, 553
- sdRead
 - Serial Driver, 555
- sdReadTimeout
 - Serial Driver, 555
- sdReadl
 - Serial Driver, 555
- sdRequestDataI
 - Serial Driver, 560
- sdStart
 - Serial Driver, 558
- sdStop
 - Serial Driver, 558
- sdWrite
 - Serial Driver, 554
- sdWriteTimeout
 - Serial Driver, 554
- sdWritel
 - Serial Driver, 554
- sd_cmd6_check_status
 - SDC Driver, 527
- sd_cmd6_construct
 - SDC Driver, 526
- sd_cmd6_extract_info
 - SDC Driver, 527
- sd_detect_bus_clk
 - SDC Driver, 527
- sd_init
 - SDC Driver, 525
- sd_lld_init
 - SDC Driver, 540
- sd_lld_read
 - SDC Driver, 544
- sd_lld_send_cmd_long_crc
 - SDC Driver, 543
- sd_lld_send_cmd_none
 - SDC Driver, 542
- sd_lld_send_cmd_short
 - SDC Driver, 542
- sd_lld_send_cmd_short_crc
 - SDC Driver, 543
- sd_lld_set_bus_mode
 - SDC Driver, 541
- sd_lld_set_data_clk
 - SDC Driver, 541
- sd_lld_start
 - SDC Driver, 540
- sd_lld_start_clk
 - SDC Driver, 541
- sd_lld_stop
 - SDC Driver, 540
- sd_lld_stop_clk
 - SDC Driver, 541
- sd_lld_sync
 - SDC Driver, 545
- sd_lld_write
 - SDC Driver, 544
- sd_set_bus_width
 - SDC Driver, 530
- sd_vmt
 - SDC Driver, 545
- sdConnect

- SDC Driver, [534](#)
- sdcDisconnect
 - SDC Driver, [535](#)
- sdcErase
 - SDC Driver, [539](#)
- sdcGetAndClearErrors
 - SDC Driver, [537](#)
- sdcGetInfo
 - SDC Driver, [539](#)
- sdcInit
 - SDC Driver, [532](#)
- sdclsCardInserted
 - SDC Driver, [522](#)
- sdclsWriteProtected
 - SDC Driver, [522](#)
- sdcObjectInit
 - SDC Driver, [533](#)
- sdcRead
 - SDC Driver, [536](#)
- sdcStart
 - SDC Driver, [533](#)
- sdcStop
 - SDC Driver, [533](#)
- sdcSync
 - SDC Driver, [538](#)
- sdcWrite
 - SDC Driver, [537](#)
- sdcbusclk_t
 - SDC Driver, [524](#)
- sdcbusmode_t
 - SDC Driver, [524](#)
- sdcflags_t
 - SDC Driver, [523](#)
- sdcmode_t
 - SDC Driver, [523](#)
- sdstate_t
 - Serial Driver, [557](#)
- sduConfigureHookI
 - Serial over USB Driver, [571](#)
- sduControl
 - Serial over USB Driver, [574](#)
- sduDataReceived
 - Serial over USB Driver, [573](#)
- sduDataTransmitted
 - Serial over USB Driver, [573](#)
- sduInit
 - Serial over USB Driver, [568](#)
- sduInterruptTransmitted
 - Serial over USB Driver, [574](#)
- sduObjectInit
 - Serial over USB Driver, [569](#)
- sduRequestsHook
 - Serial over USB Driver, [572](#)
- sduSOFHookI
 - Serial over USB Driver, [572](#)
- sduStart
 - Serial over USB Driver, [569](#)
- sduStop
 - Serial over USB Driver, [570](#)
- sduSuspendHookI
 - Serial over USB Driver, [570](#)
- sduWakeupHookI
 - Serial over USB Driver, [571](#)
- sdustate_t
 - Serial over USB Driver, [568](#)
- sectors
 - flash_descriptor_t, [753](#)
- sectors_count
 - flash_descriptor_t, [753](#)
- sectors_size
 - flash_descriptor_t, [753](#)
- send_command_R1
 - MMC over SPI Driver, [399](#)
- send_command_R3
 - MMC over SPI Driver, [400](#)
- send_hdr
 - MMC over SPI Driver, [397](#)
- sensorGetChannelNumber
 - Generic Sensor Interface, [547](#)
- sensorReadCooked
 - Generic Sensor Interface, [548](#)
- sensorReadRaw
 - Generic Sensor Interface, [547](#)
- Serial Driver, [549](#)
 - _serial_driver_data, [556](#)
 - _serial_driver_methods, [552](#)
 - default_config, [564](#)
 - PLATFORM_SERIAL_USE_USART1, [556](#)
 - SD1, [564](#)
 - SD_BREAK_DETECTED, [552](#)
 - SD_FRAMING_ERROR, [552](#)
 - SD_NOISE_ERROR, [552](#)
 - SD_OVERRUN_ERROR, [552](#)
 - SD_PARITY_ERROR, [552](#)
 - SD_QUEUE_FULL_ERROR, [552](#)
 - SD_READY, [557](#)
 - SD_STOP, [557](#)
 - SD_UNINIT, [557](#)
 - SERIAL_BUFFERS_SIZE, [552](#)
 - SERIAL_DEFAULT_BITRATE, [552](#)
 - sd_1ld_init, [563](#)
 - sd_1ld_start, [563](#)
 - sd_1ld_stop, [563](#)
 - sdAsynchronousRead, [556](#)
 - sdAsynchronousWrite, [555](#)
 - sdControl, [562](#)
 - sdGet, [553](#)
 - sdGetTimeout, [554](#)
 - sdGetWouldBlock, [561](#)
 - sdGetI, [553](#)
 - sdIncomingDataI, [559](#)
 - sdInit, [557](#)
 - sdObjectInit, [557](#)
 - sdPut, [553](#)
 - sdPutTimeout, [553](#)
 - sdPutWouldBlock, [560](#)

- sdPutI, 553
- sdRead, 555
- sdReadTimeout, 555
- sdReadI, 555
- sdRequestDataI, 560
- sdStart, 558
- sdStop, 558
- sdWrite, 554
- sdWriteTimeout, 554
- sdWriteI, 554
- sdstate_t, 557
- SerialDriver, 556
- Serial over USB Driver, 565
 - _serial_usb_driver_data, 567
 - _serial_usb_driver_methods, 567
- ibnotify, 568
- obnotify, 568
- SDU_READY, 568
- SDU_STOP, 568
- SDU_UNINIT, 568
- SERIAL_USB_BUFFERS_NUMBER, 567
- SERIAL_USB_BUFFERS_SIZE, 567
- sduConfigureHookI, 571
- sduControl, 574
- sduDataReceived, 573
- sduDataTransmitted, 573
- sduInit, 568
- sduInterruptTransmitted, 574
- sduObjectInit, 569
- sduRequestsHook, 572
- sduSOFHookI, 572
- sduStart, 569
- sduStop, 570
- sduSuspendHookI, 570
- sduWakeupHookI, 571
- sdustate_t, 568
- SerialUSBDriver, 567
- SerialConfig, 806
 - speed, 807
- SerialDriver, 807
 - Serial Driver, 556
 - vmt, 808
- SerialDriverVMT, 808
- SerialUSBConfig, 810
 - bulk_in, 811
 - bulk_out, 811
 - int_in, 811
 - usbp, 811
- SerialUSBDriver, 811
 - Serial over USB Driver, 567
 - vmt, 813
- SerialUSBDriverVMT, 813
- set_address
 - USB Driver, 662
- setup
 - USBDriver, 832
- setup_cb
 - USBEndpointConfig, 835
- size
 - flash_sector_descriptor_t, 754
 - I2SConfig, 759
 - MACReceiveDescriptor, 773
 - MACTransmitDescriptor, 773
 - mfs_data_header_t, 775
- sleep_event
 - CANDriver, 734
- sof_cb
 - USBConfig, 829
- speed
 - SerialConfig, 807
- spi_llc_abort
 - SPI Driver, 602
- spi_llc_exchange
 - SPI Driver, 601
- spi_llc_ignore
 - SPI Driver, 600
- spi_llc_init
 - SPI Driver, 599
- spi_llc_polled_exchange
 - SPI Driver, 602
- spi_llc_receive
 - SPI Driver, 602
- spi_llc_select
 - SPI Driver, 600
- spi_llc_send
 - SPI Driver, 601
- spi_llc_start
 - SPI Driver, 599
- spi_llc_stop
 - SPI Driver, 600
- spi_llc_unselect
 - SPI Driver, 600
- spiAbort
 - SPI Driver, 593
- spiAbortI
 - SPI Driver, 593
- spiAcquireBus
 - SPI Driver, 598
- spiExchange
 - SPI Driver, 595
- spiIgnore
 - SPI Driver, 594
- spiInit
 - SPI Driver, 586
- spiObjectInit
 - SPI Driver, 587
- spiPolledExchange
 - SPI Driver, 583
- spiReceive
 - SPI Driver, 597
- spiReleaseBus
 - SPI Driver, 598
- spiSelect
 - SPI Driver, 588
- spiSelectI
 - SPI Driver, 580

- spiSend
 - SPI Driver, [596](#)
- spiStart
 - SPI Driver, [587](#)
- spiStartExchange
 - SPI Driver, [590](#)
- spiStartExchangel
 - SPI Driver, [581](#)
- spiStartIgnore
 - SPI Driver, [589](#)
- spiStartIgnoreI
 - SPI Driver, [581](#)
- spiStartReceive
 - SPI Driver, [592](#)
- spiStartReceiveI
 - SPI Driver, [582](#)
- spiStartSend
 - SPI Driver, [591](#)
- spiStartSendI
 - SPI Driver, [582](#)
- spiStop
 - SPI Driver, [588](#)
- spiUnselect
 - SPI Driver, [589](#)
- spiUnselectI
 - SPI Driver, [580](#)
- spicallback_t
 - SPI Driver, [586](#)
- spip
 - MMCCConfig, [782](#)
- spistate_t
 - SPI Driver, [586](#)
- ssline
 - SPIConfig, [817](#)
- ssmask
 - SPIConfig, [817](#)
- sspad
 - SPIConfig, [817](#)
- ssport
 - SPIConfig, [817](#)
- st_llc_get_alarm
 - ST Driver, [609](#)
- st_llc_get_counter
 - ST Driver, [608](#)
- st_llc_init
 - ST Driver, [607](#)
- st_llc_is_alarm_active
 - ST Driver, [609](#)
- st_llc_set_alarm
 - ST Driver, [608](#)
- st_llc_start_alarm
 - ST Driver, [608](#)
- st_llc_stop_alarm
 - ST Driver, [608](#)
- stGetAlarm
 - ST Driver, [607](#)
- stGetCounter
 - ST Driver, [604](#)
- stInit
 - ST Driver, [605](#)
- stIsAlarmActive
 - ST Driver, [605](#)
- stSetAlarm
 - ST Driver, [606](#)
- stStartAlarm
 - ST Driver, [605](#)
- stStopAlarm
 - ST Driver, [606](#)
- state
 - ADCDriver, [695](#)
 - CANDriver, [734](#)
 - CRYDriver, [740](#)
 - DACDriver, [744](#)
 - EXTDriver, [748](#)
 - GPTDriver, [756](#)
 - I2CDriver, [758](#)
 - I2SDriver, [760](#)
 - ICUDriver, [763](#)
 - MACDriver, [771](#)
 - MFSDriver, [780](#)
 - PWMDriver, [794](#)
 - QSPIDriver, [797](#)
 - SPIDriver, [818](#)
 - UARTDriver, [823](#)
 - USBDriver, [831](#)
 - WDGDriver, [839](#)
- status
 - USBDriver, [832](#)
- streamGet
 - Abstract Streams, [258](#)
- streamPut
 - Abstract Streams, [257](#)
- streamRead
 - Abstract Streams, [257](#)
- streamWrite
 - Abstract Streams, [257](#)
- Support Code, [340](#)
- suspended
 - io_buffers_queue, [764](#)
- sync
 - MMC over SPI Driver, [401](#)
- sysinterval_t
 - OSAL, [352](#)
- syssts_t
 - OSAL, [352](#)
- system_time_t
 - OSAL, [352](#)
- TIME
 - CANRxFrame, [736](#)
- tdqueue
 - MACDriver, [772](#)
- thermometerGetChannelsNumber
 - Generic Thermometer Interface, [611](#)
- thermometerReadCooked
 - Generic Thermometer Interface, [612](#)
- thermometerReadRaw

- Generic Thermometer Interface, 611
- thermometerResetBias
 - Generic Thermometer Interface, 613
- thermometerResetSensitivity
 - Generic Thermometer Interface, 614
- thermometerSetBias
 - Generic Thermometer Interface, 612
- thermometerSetSensitivity
 - Generic Thermometer Interface, 613
- thread
 - ADCDriver, 695
 - DACDriver, 744
 - QSPIDriver, 797
 - SPIDriver, 819
 - USBInEndpointState, 837
 - USBOutEndpointState, 838
- thread_reference_t
 - OSAL, 352
- threadrx
 - UARTDriver, 823
- threads
 - palevent_t, 790
- threads_queue_t, 819
- threadtx
 - UARTDriver, 823
- top
 - io_buffers_queue, 765
- transmitting
 - USBDriver, 831
- tx_buffer
 - I2SConfig, 759
- txbuf
 - USBInEndpointState, 837
- txcnt
 - USBInEndpointState, 837
- txempty_event
 - CANDriver, 734
- txend1_cb
 - UARTConfig, 820
- txend2_cb
 - UARTConfig, 821
- txqueue
 - CANDriver, 734
- txsize
 - USBInEndpointState, 837
- txstate
 - UARTDriver, 823
- UART Driver, 615
 - _uart_rx_complete_isr_code, 624
 - _uart_rx_error_isr_code, 625
 - _uart_rx_idle_code, 625
 - _uart_timeout_isr_code, 626
 - _uart_tx1_isr_code, 623
 - _uart_tx2_isr_code, 623
 - _uart_wakeup_rx_complete_isr, 622
 - _uart_wakeup_rx_error_isr, 622
 - _uart_wakeup_rx_timeout_isr, 622
 - _uart_wakeup_tx1_isr, 620
 - _uart_wakeup_tx2_isr, 620
 - PLATFORM_UART_USE_UART1, 626
 - UART_BREAK_DETECTED, 619
 - UART_FRAMING_ERROR, 619
 - UART_NO_ERROR, 619
 - UART_NOISE_ERROR, 619
 - UART_OVERRUN_ERROR, 619
 - UART_PARITY_ERROR, 619
 - UART_READY, 627
 - UART_RX_ACTIVE, 628
 - UART_RX_COMPLETE, 628
 - UART_RX_IDLE, 628
 - UART_STOP, 627
 - UART_TX_ACTIVE, 628
 - UART_TX_COMPLETE, 628
 - UART_TX_IDLE, 628
 - UART_UNINIT, 627
 - UART_USE_MUTUAL_EXCLUSION, 620
 - UART_USE_WAIT, 620
 - UARTD1, 642
 - UARTDriver, 627
 - uart_llc_init, 640
 - uart_llc_start, 640
 - uart_llc_start_receive, 642
 - uart_llc_start_send, 641
 - uart_llc_stop, 641
 - uart_llc_stop_receive, 642
 - uart_llc_stop_send, 641
 - uartAcquireBus, 639
 - uartInit, 628
 - uartObjectInit, 628
 - uartReceiveTimeout, 638
 - uartReleaseBus, 639
 - uartSendFullTimeout, 637
 - uartSendTimeout, 636
 - uartStart, 629
 - uartStartReceive, 633
 - uartStartReceiveI, 634
 - uartStartSend, 630
 - uartStartSendI, 631
 - uartStop, 629
 - uartStopReceive, 634
 - uartStopReceiveI, 635
 - uartStopSend, 631
 - uartStopSendI, 632
 - uartcb_t, 627
 - uartccb_t, 627
 - uartccb_t, 627
 - uartflags_t, 627
 - uartrxstate_t, 628
 - uartstate_t, 627
 - uarttxstate_t, 628
- UART_BREAK_DETECTED
 - UART Driver, 619
- UART_FRAMING_ERROR
 - UART Driver, 619
- UART_NO_ERROR
 - UART Driver, 619

- UART_NOISE_ERROR
 - UART Driver, [619](#)
- UART_OVERRUN_ERROR
 - UART Driver, [619](#)
- UART_PARITY_ERROR
 - UART Driver, [619](#)
- UART_READY
 - UART Driver, [627](#)
- UART_RX_ACTIVE
 - UART Driver, [628](#)
- UART_RX_COMPLETE
 - UART Driver, [628](#)
- UART_RX_IDLE
 - UART Driver, [628](#)
- UART_STOP
 - UART Driver, [627](#)
- UART_TX_ACTIVE
 - UART Driver, [628](#)
- UART_TX_COMPLETE
 - UART Driver, [628](#)
- UART_TX_IDLE
 - UART Driver, [628](#)
- UART_UNINIT
 - UART Driver, [627](#)
- UART_USE_MUTUAL_EXCLUSION
 - Configuration, [334](#)
 - UART Driver, [620](#)
- UART_USE_WAIT
 - Configuration, [334](#)
 - UART Driver, [620](#)
- UARTConfig, [819](#)
 - rxchar_cb, [821](#)
 - rxend_cb, [821](#)
 - rxerr_cb, [821](#)
 - txend1_cb, [820](#)
 - txend2_cb, [821](#)
- UARTD1
 - UART Driver, [642](#)
- UARTDriver, [821](#)
 - config, [823](#)
 - early, [823](#)
 - mutex, [823](#)
 - rxstate, [823](#)
 - state, [823](#)
 - threadrx, [823](#)
 - threadtx, [823](#)
 - txstate, [823](#)
 - UART Driver, [627](#)
- USB CDC Header, [683](#)
- USB Driver, [643](#)
 - _usb_ep0in, [674](#)
 - _usb_ep0out, [675](#)
 - _usb_ep0setup, [673](#)
 - _usb_isr_invoke_event_cb, [656](#)
 - _usb_isr_invoke_in_cb, [657](#)
 - _usb_isr_invoke_out_cb, [657](#)
 - _usb_isr_invoke_setup_cb, [657](#)
 - _usb_isr_invoke_sof_cb, [656](#)
 - _usb_reset, [671](#)
 - _usb_suspend, [672](#)
 - _usb_wakeup, [673](#)
 - default_handler, [662](#)
 - EP_STATUS_ACTIVE, [661](#)
 - EP_STATUS_DISABLED, [661](#)
 - EP_STATUS_STALLED, [661](#)
 - ep0_state, [681](#)
 - ep0config, [682](#)
 - in, [682](#)
 - out, [682](#)
 - PLATFORM_USB_USE_USB1, [658](#)
 - set_address, [662](#)
 - USB_ACTIVE, [661](#)
 - USB_DESC_BCD, [650](#)
 - USB_DESC_BYTE, [649](#)
 - USB_DESC_CONFIGURATION_SIZE, [650](#)
 - USB_DESC_CONFIGURATION, [650](#)
 - USB_DESC_DEVICE, [650](#)
 - USB_DESC_ENDPOINT_SIZE, [652](#)
 - USB_DESC_ENDPOINT, [652](#)
 - USB_DESC_INDEX, [649](#)
 - USB_DESC_INTERFACE_ASSOCIATION_SIZE, [651](#)
 - USB_DESC_INTERFACE_ASSOCIATION, [651](#)
 - USB_DESC_INTERFACE_SIZE, [651](#)
 - USB_DESC_INTERFACE, [651](#)
 - USB_DESC_WORD, [650](#)
 - USB_EP0_ERROR, [662](#)
 - USB_EP0_IN_SENDING_STS, [662](#)
 - USB_EP0_IN_TX, [661](#)
 - USB_EP0_IN_WAITING_TX0, [662](#)
 - USB_EP0_OUT_RX, [662](#)
 - USB_EP0_OUT_WAITING_STS, [662](#)
 - USB_EP0_STATUS_STAGE, [658](#)
 - USB_EP0_STP_WAITING, [661](#)
 - USB_EP_MODE_TYPE_BULK, [652](#)
 - USB_EP_MODE_TYPE_CTRL, [652](#)
 - USB_EP_MODE_TYPE_INTR, [652](#)
 - USB_EP_MODE_TYPE_ISOC, [652](#)
 - USB_EP_MODE_TYPE, [652](#)
 - USB_EVENT_ADDRESS, [662](#)
 - USB_EVENT_CONFIGURED, [662](#)
 - USB_EVENT_RESET, [662](#)
 - USB_EVENT_STALLED, [662](#)
 - USB_EVENT_SUSPEND, [662](#)
 - USB_EVENT_UNCONFIGURED, [662](#)
 - USB_EVENT_WAKEUP, [662](#)
 - USB_MAX_ENDPOINTS, [658](#)
 - USB_READY, [661](#)
 - USB_SELECTED, [661](#)
 - USB_SET_ADDRESS_ACK_HANDLING, [658](#)
 - USB_SET_ADDRESS_MODE, [658](#)
 - USB_STOP, [661](#)
 - USB_SUSPENDED, [661](#)
 - USB_UNINIT, [661](#)
 - USB_USE_WAIT, [652](#)
 - USBD1, [681](#)

- USBDriver, 660
- usb_lld_clear_in, 681
- usb_lld_clear_out, 681
- usb_lld_connect_bus, 659
- usb_lld_disable_endpoints, 678
- usb_lld_disconnect_bus, 659
- usb_lld_get_frame_number, 658
- usb_lld_get_status_in, 678
- usb_lld_get_status_out, 678
- usb_lld_get_transaction_size, 659
- usb_lld_init, 676
- usb_lld_init_endpoint, 677
- usb_lld_prepare_receive, 679
- usb_lld_prepare_transmit, 680
- usb_lld_read_setup, 679
- usb_lld_reset, 677
- usb_lld_set_address, 677
- usb_lld_stall_in, 681
- usb_lld_stall_out, 680
- usb_lld_start, 676
- usb_lld_start_in, 680
- usb_lld_start_out, 680
- usb_lld_stop, 676
- usb_lld_wakeup_host, 659
- usbConnectBus, 653
- usbDisableEndpointsI, 666
- usbDisconnectBus, 653
- usbGetDriverStatel, 653
- usbGetFrameNumberX, 653
- usbGetReceiveStatusI, 654
- usbGetReceiveTransactionSizeX, 655
- usbGetTransmitStatusI, 654
- usbInit, 663
- usbInitEndpointI, 665
- usbObjectInit, 664
- usbReadSetup, 655
- usbReceive, 668
- usbSetupTransfer, 655
- usbStallReceiveI, 670
- usbStallTransmitI, 670
- usbStart, 664
- usbStartReceiveI, 667
- usbStartTransmitI, 667
- usbStop, 665
- usbTransmit, 669
- usbWakeupHost, 671
- usbcallback_t, 660
- usbep0state_t, 661
- usbep_t, 660
- usbepcallback_t, 660
- usbepstatus_t, 661
- usbevent_t, 662
- usbeventcb_t, 660
- usbgetdescriptor_t, 661
- usbreqhandler_t, 660
- usbstate_t, 661
- USB_ACTIVE
 - USB Driver, 661
- USB_DESC_BCD
 - USB Driver, 650
- USB_DESC_BYTE
 - USB Driver, 649
- USB_DESC_CONFIGURATION_SIZE
 - USB Driver, 650
- USB_DESC_CONFIGURATION
 - USB Driver, 650
- USB_DESC_DEVICE
 - USB Driver, 650
- USB_DESC_ENDPOINT_SIZE
 - USB Driver, 652
- USB_DESC_ENDPOINT
 - USB Driver, 652
- USB_DESC_INDEX
 - USB Driver, 649
- USB_DESC_INTERFACE_ASSOCIATION_SIZE
 - USB Driver, 651
- USB_DESC_INTERFACE_ASSOCIATION
 - USB Driver, 651
- USB_DESC_INTERFACE_SIZE
 - USB Driver, 651
- USB_DESC_INTERFACE
 - USB Driver, 651
- USB_DESC_WORD
 - USB Driver, 650
- USB_EP0_ERROR
 - USB Driver, 662
- USB_EP0_IN_SENDING_STS
 - USB Driver, 662
- USB_EP0_IN_TX
 - USB Driver, 661
- USB_EP0_IN_WAITING_TX0
 - USB Driver, 662
- USB_EP0_OUT_RX
 - USB Driver, 662
- USB_EP0_OUT_WAITING_STS
 - USB Driver, 662
- USB_EP0_STATUS_STAGE
 - USB Driver, 658
- USB_EP0_STP_WAITING
 - USB Driver, 661
- USB_EP_MODE_TYPE_BULK
 - USB Driver, 652
- USB_EP_MODE_TYPE_CTRL
 - USB Driver, 652
- USB_EP_MODE_TYPE_INTR
 - USB Driver, 652
- USB_EP_MODE_TYPE_ISOC
 - USB Driver, 652
- USB_EP_MODE_TYPE
 - USB Driver, 652
- USB_EVENT_ADDRESS
 - USB Driver, 662
- USB_EVENT_CONFIGURED
 - USB Driver, 662
- USB_EVENT_RESET
 - USB Driver, 662

- USB_EVENT_STALLED
 - USB Driver, [662](#)
- USB_EVENT_SUSPEND
 - USB Driver, [662](#)
- USB_EVENT_UNCONFIGURED
 - USB Driver, [662](#)
- USB_EVENT_WAKEUP
 - USB Driver, [662](#)
- USB_MAX_ENDPOINTS
 - USB Driver, [658](#)
- USB_READY
 - USB Driver, [661](#)
- USB_SELECTED
 - USB Driver, [661](#)
- USB_SET_ADDRESS_ACK_HANDLING
 - USB Driver, [658](#)
- USB_SET_ADDRESS_MODE
 - USB Driver, [658](#)
- USB_STOP
 - USB Driver, [661](#)
- USB_SUSPENDED
 - USB Driver, [661](#)
- USB_UNINIT
 - USB Driver, [661](#)
- USB_USE_WAIT
 - Configuration, [334](#)
 - USB Driver, [652](#)
- USBConfig, [827](#)
 - event_cb, [829](#)
 - get_descriptor_cb, [829](#)
 - requests_hook_cb, [829](#)
 - sof_cb, [829](#)
- USBID1
 - USB Driver, [681](#)
- USBDescriptor, [829](#)
 - ud_size, [830](#)
 - ud_string, [830](#)
- USBDriver, [830](#)
 - address, [832](#)
 - config, [831](#)
 - configuration, [832](#)
 - ep0endcb, [832](#)
 - ep0n, [832](#)
 - ep0next, [832](#)
 - ep0state, [832](#)
 - epc, [831](#)
 - in_params, [832](#)
 - out_params, [832](#)
 - receiving, [831](#)
 - saved_state, [832](#)
 - setup, [832](#)
 - state, [831](#)
 - status, [832](#)
 - transmitting, [831](#)
 - USB Driver, [660](#)
- USBEndpointConfig, [833](#)
 - ep_mode, [835](#)
 - in_cb, [835](#)
 - in_maxsize, [835](#)
 - in_state, [836](#)
 - out_cb, [835](#)
 - out_maxsize, [836](#)
 - out_state, [836](#)
 - setup_cb, [835](#)
- USBInEndpointState, [836](#)
 - thread, [837](#)
 - txbuf, [837](#)
 - txcnt, [837](#)
 - txsize, [837](#)
- USBOutEndpointState, [837](#)
 - rxbuf, [838](#)
 - rxcnt, [838](#)
 - rxsize, [838](#)
 - thread, [838](#)
- uart_lld_init
 - UART Driver, [640](#)
- uart_lld_start
 - UART Driver, [640](#)
- uart_lld_start_receive
 - UART Driver, [642](#)
- uart_lld_start_send
 - UART Driver, [641](#)
- uart_lld_stop
 - UART Driver, [641](#)
- uart_lld_stop_receive
 - UART Driver, [642](#)
- uart_lld_stop_send
 - UART Driver, [641](#)
- uartAcquireBus
 - UART Driver, [639](#)
- uartInit
 - UART Driver, [628](#)
- uartObjectInit
 - UART Driver, [628](#)
- uartReceiveTimeout
 - UART Driver, [638](#)
- uartReleaseBus
 - UART Driver, [639](#)
- uartSendFullTimeout
 - UART Driver, [637](#)
- uartSendTimeout
 - UART Driver, [636](#)
- uartStart
 - UART Driver, [629](#)
- uartStartReceive
 - UART Driver, [633](#)
- uartStartReceiveI
 - UART Driver, [634](#)
- uartStartSend
 - UART Driver, [630](#)
- uartStartSendI
 - UART Driver, [631](#)
- uartStop
 - UART Driver, [629](#)
- uartStopReceive
 - UART Driver, [634](#)

- uartStopReceive
 - UART Driver, [635](#)
- uartStopSend
 - UART Driver, [631](#)
- uartStopSendI
 - UART Driver, [632](#)
- uartcb_t
 - UART Driver, [627](#)
- uartccb_t
 - UART Driver, [627](#)
- uartecb_t
 - UART Driver, [627](#)
- uartflags_t
 - UART Driver, [627](#)
- uartrxstate_t
 - UART Driver, [628](#)
- uartstate_t
 - UART Driver, [627](#)
- uarttxstate_t
 - UART Driver, [628](#)
- ud_size
 - USBDescriptor, [830](#)
- ud_string
 - USBDescriptor, [830](#)
- unpacked_mmc_cid_t, [823](#)
- unpacked_mmc_csd_t, [824](#)
- unpacked_sdc_cid_t, [825](#)
- unpacked_sdc_csd_10_t, [825](#)
- unpacked_sdc_csd_20_t, [826](#)
- usb_lld_clear_in
 - USB Driver, [681](#)
- usb_lld_clear_out
 - USB Driver, [681](#)
- usb_lld_connect_bus
 - USB Driver, [659](#)
- usb_lld_disable_endpoints
 - USB Driver, [678](#)
- usb_lld_disconnect_bus
 - USB Driver, [659](#)
- usb_lld_get_frame_number
 - USB Driver, [658](#)
- usb_lld_get_status_in
 - USB Driver, [678](#)
- usb_lld_get_status_out
 - USB Driver, [678](#)
- usb_lld_get_transaction_size
 - USB Driver, [659](#)
- usb_lld_init
 - USB Driver, [676](#)
- usb_lld_init_endpoint
 - USB Driver, [677](#)
- usb_lld_prepare_receive
 - USB Driver, [679](#)
- usb_lld_prepare_transmit
 - USB Driver, [680](#)
- usb_lld_read_setup
 - USB Driver, [679](#)
- usb_lld_reset
 - USB Driver, [677](#)
- usb_lld_set_address
 - USB Driver, [677](#)
- usb_lld_stall_in
 - USB Driver, [681](#)
- usb_lld_stall_out
 - USB Driver, [680](#)
- usb_lld_start
 - USB Driver, [676](#)
- usb_lld_start_in
 - USB Driver, [680](#)
- usb_lld_start_out
 - USB Driver, [680](#)
- usb_lld_stop
 - USB Driver, [676](#)
- usb_lld_wakeup_host
 - USB Driver, [659](#)
- usbConnectBus
 - USB Driver, [653](#)
- usbDisableEndpointsI
 - USB Driver, [666](#)
- usbDisconnectBus
 - USB Driver, [653](#)
- usbGetDriverStateI
 - USB Driver, [653](#)
- usbGetFrameNumberX
 - USB Driver, [653](#)
- usbGetReceiveStatusI
 - USB Driver, [654](#)
- usbGetReceiveTransactionSizeX
 - USB Driver, [655](#)
- usbGetTransmitStatusI
 - USB Driver, [654](#)
- usbInit
 - USB Driver, [663](#)
- usbInitEndpointI
 - USB Driver, [665](#)
- usbObjectInit
 - USB Driver, [664](#)
- usbReadSetup
 - USB Driver, [655](#)
- usbReceive
 - USB Driver, [668](#)
- usbSetupTransfer
 - USB Driver, [655](#)
- usbStallReceiveI
 - USB Driver, [670](#)
- usbStallTransmitI
 - USB Driver, [670](#)
- usbStart
 - USB Driver, [664](#)
- usbStartReceiveI
 - USB Driver, [667](#)
- usbStartTransmitI
 - USB Driver, [667](#)
- usbStop
 - USB Driver, [665](#)
- usbTransmit

- USB Driver, [669](#)
- usbWakeupHost
 - USB Driver, [671](#)
- usbcallback_t
 - USB Driver, [660](#)
- usbep0state_t
 - USB Driver, [661](#)
- usbep_t
 - USB Driver, [660](#)
- usbepcallback_t
 - USB Driver, [660](#)
- usbepstatus_t
 - USB Driver, [661](#)
- usbevent_t
 - USB Driver, [662](#)
- usbeventcb_t
 - USB Driver, [660](#)
- usbgetdescriptor_t
 - USB Driver, [661](#)
- usbp
 - SerialUSBConfig, [811](#)
- usbreqhandler_t
 - USB Driver, [660](#)
- usbstate_t
 - USB Driver, [661](#)
- used_space
 - MFSDriver, [780](#)
- vmt
 - BaseAccelerometer, [697](#)
 - BaseAsynchronousChannel, [700](#)
 - BaseBarometer, [704](#)
 - BaseBlockDevice, [707](#)
 - BaseChannel, [710](#)
 - BaseCompass, [713](#)
 - BaseFlash, [716](#)
 - BaseGyroscope, [718](#)
 - BaseHygrometer, [720](#)
 - BaseObject, [722](#)
 - BaseSensor, [724](#)
 - BaseSequentialStream, [727](#)
 - BaseThermometer, [730](#)
 - FileStream, [750](#)
 - JESD215Flash, [769](#)
 - MMCDriver, [783](#)
 - MMCSDBlockDevice, [787](#)
 - RTCDriver, [800](#)
 - SDCDriver, [804](#)
 - SerialDriver, [808](#)
 - SerialUSBDriver, [813](#)
- WDG Driver, [685](#)
 - PLATFORM_WDG_USE_WDG1, [686](#)
 - WDG_READY, [686](#)
 - WDG_STOP, [686](#)
 - WDG_UNINIT, [686](#)
 - WDGDriver, [686](#)
 - wdg_ild_init, [688](#)
 - wdg_ild_reset, [689](#)
- wdg_ild_start, [689](#)
- wdg_ild_stop, [689](#)
- wdgInit, [686](#)
- wdgReset, [688](#)
- wdgResetI, [686](#)
- wdgStart, [687](#)
- wdgStop, [687](#)
- wdgstate_t, [686](#)
- WDG_READY
 - WDG Driver, [686](#)
- WDG_STOP
 - WDG Driver, [686](#)
- WDG_UNINIT
 - WDG Driver, [686](#)
- WDGConfig, [838](#)
- WDGDriver, [839](#)
 - config, [839](#)
 - state, [839](#)
 - WDG Driver, [686](#)
- wait
 - MMC over SPI Driver, [396](#)
- waiting
 - io_buffers_queue, [764](#)
- wakeup_event
 - CANDriver, [734](#)
- wdg_ild_init
 - WDG Driver, [688](#)
- wdg_ild_reset
 - WDG Driver, [689](#)
- wdg_ild_start
 - WDG Driver, [689](#)
- wdg_ild_stop
 - WDG Driver, [689](#)
- wdgInit
 - WDG Driver, [686](#)
- wdgReset
 - WDG Driver, [688](#)
- wdgResetI
 - WDG Driver, [686](#)
- wdgStart
 - WDG Driver, [687](#)
- wdgStop
 - WDG Driver, [687](#)
- wdgstate_t
 - WDG Driver, [686](#)
- width_cb
 - ICUConfig, [762](#)
- year
 - RTCDateTime, [799](#)