

MicroLabBox

RTLib Reference

Release 2021-A – May 2021

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: <http://www.dspace.com/go/locations>
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: <http://www.dspace.com/go/supportrequest>. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit <http://www.dspace.com/go/patches> for software updates and patches.

Important Notice

This publication contains proprietary information that is protected by copyright. All rights are reserved. The publication may be printed for personal or internal use provided all the proprietary markings are retained on all printed copies. In all other cases, the publication must not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of dSPACE GmbH.

© 2014 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

This publication and the contents hereof are subject to change without notice.

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

Contents

About This Reference	23
Basic Functions	25
Data Types and Definitions.....	26
Elementary Data Types.....	26
Initialization.....	28
init.....	28
Background Service.....	29
RTLIB_BACKGROUND_SERVICE.....	29
rtlib_background_hook.....	29
Time Interval Measurement.....	32
Data Types for Time Measurement.....	33
Example of Using Time Measurement Functions.....	34
srtk_tic_continue.....	34
srtk_tic_count.....	35
srtk_tic_delay.....	36
srtk_tic_diff.....	37
srtk_tic_elapsed.....	38
srtk_tic_halt.....	39
srtk_tic_read.....	40
srtk_tic_start.....	41
srtk_tic_total_read.....	41
srtk_timebase_fltread.....	42
srtk_timebase_low_read.....	42
srtk_timebase_read.....	43
RTLIB_TIC_CONTINUE.....	44
RTLIB_TIC_COUNT.....	44
RTLIB_TIC_DELAY.....	45
RTLIB_TIC_DIFF.....	46
RTLIB_TIC_ELAPSED.....	47
RTLIB_TIC_HALT.....	48
RTLIB_TIC_READ.....	49
RTLIB_TIC_READ_TOTAL.....	50
RTLIB_TIC_START.....	50

Time-Stamping.....	52
General Information on Time-Stamping.....	52
Basic Principles of Time-Stamping.....	52
Principles of an Absolute Time in Single-Processor and Multiprocessor Systems.....	53
Data Types and Global Variables for Time-Stamping.....	54
Data Types Used for Time-Stamping.....	54
Time-Stamping Functions.....	54
ts_init.....	55
ts_mat_period_get.....	57
ts_mit_period_get.....	57
ts_reset.....	58
ts_time_read.....	58
ts_timestamp_read.....	59
ts_timestamp_compare.....	60
ts_timestamp_interval.....	61
ts_time_offset.....	61
ts_timestamp_offset.....	62
ts_time_calculate.....	63
ts_timestamp_calculate.....	64
Timer A.....	65
Example of Using Timer A Functions.....	65
RTLIB_SRT_PERIOD.....	66
srtk_timerA_period_set.....	67
srtk_timerA_period_reload.....	67
srtk_timerA_read.....	68
srtk_timerA_start.....	69
srtk_timerA_stop.....	69
Timer B.....	71
Example of Using Timer B Functions.....	72
srtk_timerB_init.....	72
srtk_timerB_compare_set.....	73
srtk_timerB_compare_set_periodically.....	74
srtk_timerB_read.....	75
srtk_timerB_start.....	75
srtk_timerB_stop.....	76
Timer D.....	77
Example of Using Timer D functions.....	77
srtk_timerD_period_set.....	78

srtk_timerD_period_reload.....	79
srtk_timerD_read.....	79
srtk_timerD_start.....	80
srtk_timerD_stop.....	81
Timer Interrupt Control.....	82
srtk_begin_isr_timerA.....	83
srtk_begin_isr_timerB.....	84
srtk_begin_isr_timerD.....	85
srtk_end_isr_timerA.....	85
srtk_end_isr_timerB.....	86
srtk_end_isr_timerD.....	86
srtk_start_isr_timerA.....	87
srtk_start_isr_timerB.....	88
srtk_start_isr_timerD.....	89
RTLIB_SRT_ISR_BEGIN.....	90
RTLIB_SRT_ISR_END.....	91
RTLIB_SRT_START.....	91
Interrupt Handling.....	93
srtk_disable_hardware_int.....	94
srtk_disable_hardware_int_bm.....	95
srtk_enable_hardware_int.....	96
srtk_enable_hardware_int_bm.....	97
srtk_get_interrupt_flag.....	99
srtk_get_interrupt_flag_bm.....	100
srtk_get_interrupt_vector.....	101
srtk_reset_interrupt_flag.....	102
srtk_reset_interrupt_flag_bm.....	103
srtk_set_interrupt_vector.....	104
RTLIB_INT_DISABLE.....	106
RTLIB_INT_ENABLE.....	106
RTLIB_INT_RESTORE.....	107
RTLIB_INT_SAVE_AND_DISABLE.....	107
RTLIB_SRT_DISABLE.....	108
RTLIB_SRT_ENABLE.....	109
Subinterrupt Handling.....	110
Basic Principles of Subinterrupt Handling.....	111
Example of Using a Subinterrupt Sender.....	111
Example of Using a Subinterrupt Handler.....	112
Example of Using a Subinterrupt Receiver.....	113
Data Types for Subinterrupt Handling.....	115
dssint_define_int_sender.....	116

dssint_define_int_sender_1.....	118
dssint_define_int_receiver.....	120
dssint_define_int_receiver_1.....	122
dssint_subint_disable.....	124
dssint_subint_enable.....	125
dssint_interrupt.....	126
dssint_decode.....	126
dssint_acknowledge.....	127
dssint_subint_reset.....	128
 Message Handling.....	130
Basic Principles of Message Handling.....	131
Data Types and Symbols for Message Handling.....	132
msg_error_set.....	134
msg_warning_set.....	135
msg_info_set.....	135
msg_set.....	136
msg_error_printf.....	138
msg_warning_printf.....	140
msg_info_printf.....	141
msg_printf.....	142
msg_default_dialog_set.....	144
msg_mode_set.....	145
msg_reset.....	146
msg_last_error_number.....	146
msg_last_error_submodule.....	147
msg_error_clear.....	149
msg_error_hook_set.....	150
msg_init.....	151
 System Functions.....	153
Buzzer Control.....	153
Buzzer_apply.....	154
Buzzer_create.....	155
Buzzer_setDuration.....	156
Buzzer_setFrequency.....	157
Buzzer_setNumberOfBeeps.....	158
Buzzer_setPause.....	160
Buzzer_start.....	161
Buzzer_write.....	161
 LED Control.....	162
Led_apply.....	163

Led_create.....	164
Led_setBlueColor.....	165
Led_setGreenColor.....	167
Led_setRedColor.....	168
Led_start.....	169
Led_write.....	170
Sensor Supply.....	171
SensorSupply_apply.....	171
SensorSupply_create.....	173
SensorSupply_setSensorState.....	174
SensorSupply_setVoltage.....	175
SensorSupply_start.....	177
SensorSupply_write.....	178
Special Processor Functions.....	180
RTLIB_FORCE_IN_ORDER.....	180
RTLIB_SYNC.....	181
Conversion Functions.....	182
RTLIB_CONV_FLOAT32_FROM_IEEE32.....	182
RTLIB_CONV_FLOAT32_FROM_TI32.....	183
RTLIB_CONV_FLOAT32_TO_IEEE32.....	183
RTLIB_CONV_FLOAT_TO_SATURATED_INT32.....	184
RTLIB_CONV_FLOAT32_TO_TI32.....	184
Standard Macros.....	186
init().....	187
RTLIB_INIT.....	188
RTLIB_TERMINATE.....	189
RTLIB_GET_SERIAL_NUMBER().....	189
RTLIB_REGISTER_BACKGROUND_HANDLER.....	190
RTLIB_REGISTER_TERMINATE_HANDLER.....	191
RTLIB_REGISTER_UNLOAD_HANDLER.....	191
RTLIB_MALLOC_PROT.....	192
RTLIB_CALLOC_PROT.....	193
RTLIB_REALLOC_PROT.....	194
RTLIB_FREE_PROT.....	194
I/O Functions	197
Common Concepts for Implementing I/O Functions.....	198
Implementing I/O Functions.....	198

A/D Conversion.....	202
Analog In Class 1.....	202
AdcCl1AnalogIn_abortBurst.....	204
AdcCl1AnalogIn_abortBurstMultiple.....	205
AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_disableInterrupts.....	209
AdcCl1AnalogIn_enableInterrupts.....	211
AdcCl1AnalogIn_getBurstCurrent.....	212
AdcCl1AnalogIn_getBurstImmediate.....	214
AdcCl1AnalogIn_getFailureInfo.....	216
AdcCl1AnalogIn_getSingleValue.....	218
AdcCl1AnalogIn_isDataReady.....	219
AdcCl1AnalogIn_read.....	221
AdcCl1AnalogIn_readFailureInfo.....	222
AdcCl1AnalogIn_setBurstMode.....	223
AdcCl1AnalogIn_setBurstSize.....	225
AdcCl1AnalogIn_setBurstTrigger.....	226
AdcCl1AnalogIn_setChannelState.....	228
AdcCl1AnalogIn_setConversionMode.....	229
AdcCl1AnalogIn_setConversTrigger.....	231
AdcCl1AnalogIn_setInterruptMode.....	233
AdcCl1AnalogIn_setIoIntVector.....	234
AdcCl1AnalogIn_setTimerPeriod.....	236
AdcCl1AnalogIn_setTriggerLineIn.....	237
AdcCl1AnalogIn_setTriggerLineOut.....	238
AdcCl1AnalogIn_setTriggerLineOutStatus.....	240
AdcCl1AnalogIn_setTriggerMode.....	241
AdcCl1AnalogIn_start.....	243
AdcCl1AnalogIn_stop.....	244
AdcCl1AnalogIn_triggerBurst.....	245
AdcCl1AnalogIn_triggerBurstMultiple.....	246
AdcCl1AnalogIn_triggerConversion.....	247
AdcCl1AnalogIn_triggerConversionMultiple.....	248
AdcCl1AnalogIn_write.....	249
Analog In Class 2.....	250
AdcCl2AnalogIn_apply.....	251
AdcCl2AnalogIn_create.....	252
AdcCl2AnalogIn_getInputValue.....	254
AdcCl2AnalogIn_read.....	255

AdcCl2AnalogIn_start.....	256
Example of Implementing A/D Conversion Using ADC Class 2.....	257
D/A Conversion.....	259
Analog Out Class 1.....	259
DacCl1AnalogOut_apply.....	260
DacCl1AnalogOut_create.....	261
DacCl1AnalogOut_setOutputValue.....	263
DacCl1AnalogOut_start.....	264
DacCl1AnalogOut_write.....	266
Example of Implementing D/A Conversion Using DAC Class 1.....	267
Bit I/O.....	268
Digital In Class 1.....	268
DioCl1DigIn_apply.....	269
DioCl1DigIn_create.....	270
DioCl1DigIn_disableInterrupts.....	272
DioCl1DigIn_enableInterrupts.....	273
DioCl1DigIn_getChMaskInData.....	274
DioCl1DigIn_getPortInData.....	275
DioCl1DigIn_read.....	276
DioCl1DigIn_setEventDelay.....	278
DioCl1DigIn_setEventDownSample.....	279
DioCl1DigIn_setInputFilter.....	280
DioCl1DigIn_setInterruptMode.....	281
DioCl1DigIn_setIoIntVector.....	282
DioCl1DigIn_setTriggerLineOut.....	284
DioCl1DigIn_setTriggerLineOutStatus.....	285
DioCl1DigIn_setTriggerMode.....	286
DioCl1DigIn_start.....	288
Example of Implementing Bit I/O Inputs Using DIO Class 1.....	289
Digital Out Class 1.....	290
DioCl1DigOut_apply.....	292
DioCl1DigOut_create.....	293
DioCl1DigOut_disableInterrupts.....	294
DioCl1DigOut_enableInterrupts.....	295
DioCl1DigOut_setChMaskOutData.....	297
DioCl1DigOut_setChMaskOutHighZ.....	298
DioCl1DigOut_setEventDelay.....	299
DioCl1DigOut_setEventDownsample.....	300
DioCl1DigOut_setInterruptMode.....	301
DioCl1DigOut_setIoIntVector.....	303

DioCl1DigOut_setPortOutData.....	304
DioCl1DigOut_setPortOutHighZ.....	305
DioCl1DigOut_setRisingEdgeDelay.....	307
DioCl1DigOut_setSignalVoltage.....	308
DioCl1DigOut_setTriggerLineOut.....	309
DioCl1DigOut_setTriggerLineOutStatus.....	310
DioCl1DigOut_setTriggerMode.....	311
DioCl1DigOut_start.....	313
DioCl1DigOut_write.....	314
Example of Implementing Bit I/O Outputs Using DIO Class 1.....	315
 Timing I/O.....	317
PWM Signal Generation (PWM Out).....	317
DioCl1PwmOut_apply.....	319
DioCl1PwmOut_create.....	320
DioCl1PwmOut_disableInterrupts.....	321
DioCl1PwmOut_enableInterrupts.....	322
DioCl1PwmOut_setDutyCycle.....	324
DioCl1PwmOut_setEventDelay.....	325
DioCl1PwmOut_setEventDownsample.....	326
DioCl1PwmOut_setInterruptMode.....	327
DioCl1PwmOut_setInvertingMode.....	328
DioCl1PwmOut_setIoIntVector.....	329
DioCl1PwmOut_setOutputHighZ.....	330
DioCl1PwmOut_setPeriod.....	332
DioCl1PwmOut_setRisingEdgeDelay.....	333
DioCl1PwmOut_setSignalVoltage.....	334
DioCl1PwmOut_setTriggerLineOut.....	335
DioCl1PwmOut_setTriggerLineOutStatus.....	336
DioCl1PwmOut_setTriggerMode.....	338
DioCl1PwmOut_setUpdateMode.....	339
DioCl1PwmOut_start.....	340
DioCl1PwmOut_write.....	341
 PWM Signal Measurement (PWM In).....	342
DioCl1Pwmln_apply.....	344
DioCl1Pwmln_create.....	345
DioCl1Pwmln_disableInterrupts.....	346
DioCl1Pwmln_enableInterrupts.....	347
DioCl1Pwmln_getDutyCycle.....	349
DioCl1Pwmln_getFrequency.....	350
DioCl1Pwmln_read.....	351
DioCl1Pwmln_setEventDelay.....	352

DioCl1Pwmln_setEventDownsample.....	353
DioCl1Pwmln_setInterruptMode.....	354
DioCl1Pwmln_setIoIntVector.....	356
DioCl1Pwmln_setTimeout.....	357
DioCl1Pwmln_setTriggerMode.....	358
DioCl1Pwmln_setTriggerLineOut.....	360
DioCl1Pwmln_setTriggerLineOutStatus.....	361
DioCl1Pwmln_setUpdateMode.....	362
DioCl1Pwmln_start.....	363
 Pulse Signal Generation (Pulse Out).....	364
DioCl1PulseOut_apply.....	365
DioCl1PulseOut_create.....	366
DioCl1PulseOut_setFirePulse.....	368
DioCl1PulseOut_setInvertingMode.....	369
DioCl1PulseOut_setOutputHighZ.....	370
DioCl1PulseOut_setPulseWidth.....	371
DioCl1PulseOut_setSignalVoltage.....	372
DioCl1PulseOut_setTriggerLineIn.....	373
DioCl1PulseOut_start.....	374
DioCl1PulseOut_write.....	375
 Pulse Width Measurement (Pulse In).....	376
DioCl1PulseIn_apply.....	378
DioCl1PulseIn_create.....	379
DioCl1PulseIn_disableInterrupts.....	380
DioCl1PulseIn_enableInterrupts.....	381
DioCl1PulseIn_getPulseWidth.....	382
DioCl1PulseIn_read.....	384
DioCl1PulseIn_setEdgePolarity.....	385
DioCl1PulseIn_setEventDelay.....	386
DioCl1PulseIn_setInterruptMode.....	387
DioCl1PulseIn_setIoIntVector.....	388
DioCl1PulseIn_setTriggerMode.....	390
DioCl1PulseIn_setTriggerLineOut.....	391
DioCl1PulseIn_setTriggerLineOutStatus.....	392
DioCl1PulseIn_setWidthMax.....	393
DioCl1PulseIn_start.....	394
 USB Flight Recorder	397
Introduction to the USB Flight Recorder.....	397

Nonvolatile Data Handling (NVDATA)	399
NvData_apply.....	400
NvData_create.....	401
NvData_createDataSet.....	402
NvData_read.....	403
NvData_setDimension.....	404
NvData_setName.....	405
NvData_setType.....	407
NvData_write.....	408
Example of Implementing Access to the Nonvolatile Data.....	409
Serial Interface Communication	411
Basic Principles of Serial Communication.....	412
Software FIFO Buffer.....	412
Trigger Levels.....	413
How to Handle Subinterrupts in Serial Communication.....	413
Example of a Serial Interface Communication.....	415
Data Types for Serial Communication.....	417
dsser_ISR.....	417
dsser_LSR.....	419
dsser_MSR.....	420
dsser_subint_handler_t.....	421
dsserChannel.....	422
Generic Serial Interface Communication Functions.....	424
dsser_init.....	425
dsser_free.....	426
dsser_config.....	427
dsser_transmit.....	430
dsser_receive.....	432
dsser_receive_term.....	433
dsser_fifo_reset.....	435
dsser_enable.....	436
dsser_disable.....	436
dsser_error_read.....	437
dsser_transmit_fifo_level.....	438
dsser_receive_fifo_level.....	439
dsser_status_read.....	440
dsser_handle_get.....	441
dsser_set.....	442

dsser_subint_handler_inst.....	443
dsser_subint_enable.....	444
dsser_subint_disable.....	445
dsser_word2bytes.....	446
dsser_bytes2word.....	448

Serial Peripheral Interface (SPI) 451

DioCl1Spi_apply.....	453
DioCl1Spi_create.....	454
DioCl1Spi_disableInterrupts.....	456
DioCl1Spi_enableInterrupts.....	457
DioCl1Spi_setChipSelectPolarity.....	459
DioCl1Spi_setInterruptMode.....	460
DioCl1Spi_setIoIntVector.....	461
DioCl1Spi_setOutputsHighZ.....	462
DioCl1Spi_setSignalVoltage.....	463
DioCl1Spi_setTriggerMode.....	464
DioCl1Spi_setTriggerLineOut.....	466
DioCl1Spi_setTriggerLineOutStatus.....	467
DioCl1Spi_start.....	468
DioCl1Spi_write.....	469
DioCl1SpiCycle_apply.....	470
DioCl1SpiCycle_create.....	471
DioCl1SpiCycle_getCycleData.....	473
DioCl1SpiCycle_read.....	475
DioCl1SpiCycle_setBaudRate.....	476
DioCl1SpiCycle_setBitDirection.....	477
DioCl1SpiCycle_setChipSelectConf.....	478
DioCl1SpiCycle_setClockPolarity.....	479
DioCl1SpiCycle_setClockPhase.....	480
DioCl1SpiCycle_setCycleData.....	482
DioCl1SpiCycle_setTimeAfterTran.....	483
DioCl1SpiCycle_setTimeBeforeTran.....	484
DioCl1SpiCycle_setTimeBetwWords.....	486
DioCl1SpiCycle_setTimeCsInactive.....	487
DioCl1SpiCycle_start.....	488
DioCl1SpiCycle_write.....	489

Electric Motor Control Functions 491

Angle Computation Unit (ACU).....	493
Acu_apply.....	495
Acu_create.....	496
Acu_disableInterrupts.....	498
Acu_enableInterrupts.....	499
Acu_getCurrentInputSensorNo.....	501
Acu_getElecPosition.....	502
Acu_getIsElecPosValid.....	503
Acu_getIsMechPosValid.....	504
Acu_getIsMotorSpeedValid.....	505
Acu_getIsSensorElecPosValid.....	506
Acu_getMechPosition.....	508
Acu_getMotorSpeed.....	509
Acu_getSensorElecPosition.....	510
Acu_read.....	511
Acu_setAbsolutePositionForEvent.....	513
Acu_setInputSensor.....	514
Acu_setInterruptMode.....	516
Acu_setIoIntVector.....	518
Acu_setMinMotorSpeed.....	519
Acu_setMotorPolePairCount.....	520
Acu_setPositionEventBase.....	521
Acu_setPositionEventHysteresis.....	523
Acu_setRelativePositionForEvent.....	524
Acu_setRelativePositionOffset.....	525
Acu_setSensorForPositionEvent.....	526
Acu_setSensorPolePairCount.....	527
Acu_setTriggerLineOut.....	529
Acu_setTriggerLineOutStatus.....	530
Acu_setTriggerMode.....	531
Acu_start.....	533
Acu_write.....	534
EnDat Interface.....	536
DioCl2EnDatIn_apply.....	537
DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_getConfigErr.....	540
DioCl2EnDatIn_getCrcErrorCount.....	542
DioCl2EnDatIn_getIsPositionValid.....	543
DioCl2EnDatIn_getIsRevolNoValid.....	544

DioCl2EnDatIn_getMechPosition.....	546
DioCl2EnDatIn_getRevolutionNo.....	548
DioCl2EnDatIn_getTransmissionErr.....	549
DioCl2EnDatIn_read.....	551
DioCl2EnDatIn_setClockFrequency.....	552
DioCl2EnDatIn_setCrcErrorLimit.....	555
DioCl2EnDatIn_setMultiturnRes.....	556
DioCl2EnDatIn_setNumberOfSteps.....	558
DioCl2EnDatIn_setPositionOffset.....	559
DioCl2EnDatIn_setReverseDirection.....	560
DioCl2EnDatIn_setSampleSyncEvSrc.....	562
DioCl2EnDatIn_setSingleturnRes.....	563
DioCl2EnDatIn_start.....	565
DioCl2EnDatIn_stop.....	566
 Hall Sensor Interface.....	568
Hall Sensor Class 1.....	568
DioCl1HallIn_apply.....	569
DioCl1HallIn_create.....	570
DioCl1HallIn_getIsPositionValid.....	573
DioCl1HallInGetPosition.....	574
DioCl1HallIn_getSensorSignals	576
DioCl1HallIn_read.....	577
DioCl1HallIn_setInputFilter.....	578
DioCl1HallIn_setPositionOffset.....	580
DioCl1HallIn_setReverseDirection.....	581
DioCl1HallIn_setSigEdgePositions.....	582
DioCl1HallIn_start.....	583
DioCl1HallIn_write.....	584
Hall Sensor Class 2.....	586
DioCl2HallIn_apply.....	587
DioCl2HallIn_create.....	588
DioCl2HallIn_getIsPositionValid.....	590
DioCl2HallInGetPosition.....	592
DioCl2HallIn_getSensorSignals	593
DioCl2HallIn_read.....	594
DioCl2HallIn_setInputFilter.....	596
DioCl2HallIn_setPositionOffset.....	597
DioCl2HallIn_setReverseDirection.....	598
DioCl2HallIn_setSigEdgePositions.....	599
DioCl2HallIn_start.....	601
DioCl2HallIn_write.....	602

Incremental Encoder Interface.....	604
Incremental Encoder Class 1.....	604
DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_getEncPosition.....	609
DioCl1EncoderIn_getEncVelocity.....	611
DioCl1EncoderIn_getIsIndexRaised.....	612
DioCl1EncoderIn_getMechPosition.....	613
DioCl1EncoderIn_getMechVelocity.....	614
DioCl1EncoderIn_read.....	616
DioCl1EncoderIn_setEncoderLines.....	617
DioCl1EncoderIn_setEncPosition.....	618
DioCl1EncoderIn_setEncPosValidity.....	619
DioCl1EncoderIn_setGatedMode.....	621
DioCl1EncoderIn_setIndexMode.....	622
DioCl1EncoderIn_setIndexPosition.....	623
DioCl1EncoderIn_setInputFilter.....	624
DioCl1EncoderIn_setMaxPosition.....	626
DioCl1EncoderIn_setMeasurementInterval.....	627
DioCl1EncoderIn_setMinEncVelocity.....	628
DioCl1EncoderIn_setMinMechVelocity.....	629
DioCl1EncoderIn_setMinPosition.....	630
DioCl1EncoderIn_start.....	632
DioCl1EncoderIn_stop.....	633
DioCl1EncoderIn_write.....	634
Incremental Encoder Class 2.....	635
DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_getEncPosition.....	640
DioCl2EncoderIn_getEncVelocity.....	642
DioCl2EncoderIn_getIsIndexRaised.....	643
DioCl2EncoderIn_getMechPosition.....	644
DioCl2EncoderIn_getMechVelocity.....	645
DioCl2EncoderIn_read.....	647
DioCl2EncoderIn_setEncoderLines.....	648
DioCl2EncoderIn_setEncPosition.....	649
DioCl2EncoderIn_setEncPosValidity.....	650
DioCl2EncoderIn_setGatedMode.....	652
DioCl2EncoderIn_setIndexMode.....	653
DioCl2EncoderIn_setIndexPosition.....	654
DioCl2EncoderIn_setInputFilter.....	655

DioCl2EncoderIn_setMaxPosition.....	657
DioCl2EncoderIn_setMeasurementInterval.....	658
DioCl2EncoderIn_setMinEncVelocity.....	659
DioCl2EncoderIn_setMinMechVelocity.....	660
DioCl2EncoderIn_setMinPosition.....	661
DioCl2EncoderIn_start.....	663
DioCl2EncoderIn_stop.....	664
DioCl2EncoderIn_write.....	665
 Resolver Interface.....	667
ResolverIn_apply.....	668
ResolverIn_create.....	669
ResolverIn_getFaultStatus.....	671
ResolverIn_getIsDataValid.....	673
ResolverIn_getPosition.....	674
ResolverIn_read.....	675
ResolverIn_setExcitationFreq.....	677
ResolverIn_setExcitationVoltRms.....	678
ResolverIn_setMaximumSpeed.....	679
ResolverIn_setPositionOffset.....	680
ResolverIn_setReverseDirection.....	682
ResolverIn_setSineCosineVoltRms.....	683
ResolverIn_start.....	684
 SSI Interface.....	686
DioCl2Ssiln_apply.....	687
DioCl2Ssiln_create.....	689
DioCl2Ssiln_getCommunicationErr.....	691
DioCl2Ssiln_getIsPositionValid.....	692
DioCl2Ssiln_getIsRevoltNoValid.....	694
DioCl2Ssiln_getMechPosition.....	695
DioCl2Ssiln_getRevolutionNo.....	697
DioCl2Ssiln_getSsiFrameData.....	698
DioCl2Ssiln_getTransmitErrCount.....	700
DioCl2Ssiln_read.....	701
DioCl2Ssiln_setClockFrequency.....	703
DioCl2Ssiln_setClockPause.....	705
DioCl2Ssiln_setCodeType.....	706
DioCl2Ssiln_setMultiturnRes.....	707
DioCl2Ssiln_setNoOfFirstPosBit.....	709
DioCl2Ssiln_setNumberOfSteps.....	710
DioCl2Ssiln_setPositionOffset.....	712
DioCl2Ssiln_setRepeatReading.....	713

DioCl2Ssiln_setReverseDirection.....	715
DioCl2Ssiln_setSampleSyncEvSrc.....	716
DioCl2Ssiln_setSingleturnRes.....	718
DioCl2Ssiln_setTransmitBitCount.....	719
DioCl2Ssiln_setTransmitErrLimit.....	721
DioCl2Ssiln_start.....	722
DioCl2Ssiln_stop.....	723
 Block-Commutated PWM Generation.....	725
DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_disableInterrupts.....	731
DioCl1BcPwmOut_enableInterrupts.....	732
DioCl1BcPwmOut_setAcu.....	734
DioCl1BcPwmOut_setDirection.....	735
DioCl1BcPwmOut_setDutyCycle.....	737
DioCl1BcPwmOut_setEventDelay.....	738
DioCl1BcPwmOut_setEventDownsample.....	739
DioCl1BcPwmOut_setInterruptMode.....	741
DioCl1BcPwmOut_setIoIntVector.....	742
DioCl1BcPwmOut_setOutputHighZ.....	744
DioCl1BcPwmOut_setOutputMode.....	745
DioCl1BcPwmOut_setPeriod.....	747
DioCl1BcPwmOut_setPosition.....	748
DioCl1BcPwmOut_setPositionOffset.....	749
DioCl1BcPwmOut_setRisingEdgeDelay.....	751
DioCl1BcPwmOut_setSectorCount.....	752
DioCl1BcPwmOut_setSectorPositions.....	753
DioCl1BcPwmOut_setSectorSignals.....	754
DioCl1BcPwmOut_setSignalPairCheck.....	757
DioCl1BcPwmOut_setSignalVoltage.....	758
DioCl1BcPwmOut_setStationarySignal.....	759
DioCl1BcPwmOut_setTriggerLineOut.....	761
DioCl1BcPwmOut_setTriggerLineOutStatus.....	763
DioCl1BcPwmOut_setTriggerMode.....	764
DioCl1BcPwmOut_setUpdateMode.....	766
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_stop.....	768
DioCl1BcPwmOut_write.....	769
 Multichannel PWM Signal Generation.....	771
DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_create.....	774

DioCl1MultiPwmOut_disableInterrupts.....	776
DioCl1MultiPwmOut_enableInterrupts.....	777
DioCl1MultiPwmOut_setAlignmentMode.....	779
DioCl1MultiPwmOut_setDutyCycle.....	780
DioCl1MultiPwmOut_setEventDelay.....	781
DioCl1MultiPwmOut_setEventDownsample.....	783
DioCl1MultiPwmOut_setInterruptMode.....	784
DioCl1MultiPwmOut_setInvertingMode.....	785
DioCl1MultiPwmOut_setIoIntVector.....	787
DioCl1MultiPwmOut_setOutputHighZ.....	788
DioCl1MultiPwmOut_setPeriod.....	790
DioCl1MultiPwmOut_setRisingEdgeDelay.....	791
DioCl1MultiPwmOut_setSignalVoltage.....	792
DioCl1MultiPwmOut_setTriggerLineOut.....	794
DioCl1MultiPwmOut_setTriggerLineOutStatus.....	795
DioCl1MultiPwmOut_setTriggerMode.....	796
DioCl1MultiPwmOut_setUpdateMode.....	798
DioCl1MultiPwmOut_start.....	799
DioCl1MultiPwmOut_write.....	801
Slave CAN Access Functions	803
Basics on Slave CAN Access Functions.....	804
Basic Principles of Master-Slave Communication.....	804
CAN Error Message Types.....	805
Data Structures for CAN.....	807
can_tp1_canChannel.....	807
can_tp1_canService.....	809
can_tp1_canMsg.....	812
Initialization.....	816
can_tp1_communication_init.....	816
CAN Channel Handling.....	818
can_tp1_channel_init.....	818
can_tp1_channel_start.....	821
can_tp1_channel_all_sleep.....	822
can_tp1_channel_all_wakeup.....	823
can_tp1_channel_BOff_go.....	824
can_tp1_channel_BOff_return.....	825
can_tp1_channel_set.....	826
can_tp1_channel_txqueue_clear.....	828

CAN Message Handling.....	829
can_tp1_msg_tx_register.....	830
can_tp1_msg_rx_register.....	834
can_tp1_msg_rqtx_register.....	837
can_tp1_msg_rqrq_register.....	841
can_tp1_msg_rm_register.....	844
can_tp1_msg_set.....	847
can_tp1_msg_rqtx_activate.....	849
can_tp1_msg_write.....	850
can_tp1_msg_send.....	852
can_tp1_msg_send_id.....	853
can_tp1_msg_queue_level.....	855
can_tp1_msg_txqueue_init.....	855
can_tp1_msg_send_id_queued.....	857
can_tp1_msg_txqueue_level_read.....	859
can_tp1_msg_sleep.....	860
can_tp1_msg_wakeup.....	861
can_tp1_msg_read.....	862
can_tp1_msg_trigger.....	864
can_tp1_msg_clear.....	865
can_tp1_msg_processed_register.....	866
can_tp1_msg_processed_request.....	867
can_tp1_msg_processed_read.....	868
CAN Service Functions.....	870
can_tp1_service_register.....	870
can_tp1_service_request.....	872
can_tp1_service_read.....	873
CAN Subinterrupt Handling.....	875
Defining a Callback Function.....	875
can_tp1_subint_handler_install.....	876
Utilities.....	877
can_tp1_all_data_clear.....	877
can_tp1_error_read.....	878
Examples of Using CAN.....	880
Example of Handling Transmit and Receive Messages.....	880
Example of Handling Request and Remote Messages.....	881
Example of Using Subinterrupts.....	883
Example of Using Service Functions.....	885
Example of Receiving Different Message IDs.....	886

FPGA Module Access Functions	889
FPGA Initialization.....	890
IoFpga_init.....	890
Identification Functions.....	892
IoFpga_get_board_rev.....	892
IoFpga_get_fw_id.....	893
IoFpga_get_appl_id.....	894
IoFpga_get_appl_id_string.....	895
Interrupt Functions.....	896
IoFpga_enable_int.....	896
IoFpga_disable_int.....	897
IoFpga_ack_int.....	898
IoFpga_register_isr.....	899
Data Exchange Functions.....	901
IoFpga_read_reg.....	902
IoFpga_read_reg64.....	903
IoFpga_write_reg.....	905
IoFpga_write_reg64.....	906
IoFpga_read_reg_grp.....	908
IoFpga_read_reg_grp_mixed.....	910
IoFpga_write_reg_grp.....	912
IoFpga_write_reg_grp_mixed.....	914
IoFpga_read_buf.....	917
IoFpga_read_buf64.....	919
IoFpga_write_buf.....	921
IoFpga_write_buf64.....	923
Host Programs	925
Host Settings.....	926
Compiler and C Run-Time Libraries.....	926
Environment Variables and Paths.....	926
Folder Structure.....	927
DS1202 Real-Time Library.....	927
File Extensions.....	927
Compiling, Linking and Downloading an Application.....	929
Down1202.exe.....	930
DsBuildApplication.mk.....	933
DsBuildLoad.mk.....	934

DsBuildTemplate.mk.....	934
Integrating C++ Code.....	935
Debugging an Application.....	937
ntoppc-objdump.....	937
Index	941

About This Reference

Content

This RTLib Reference (Real-Time Library) gives detailed descriptions of the C functions needed to program MicroLabBox. Internally, MicroLabBox consists of the two boards DS1202 and DS1302. The DS1202 board provides the basic system and I/O capabilities. The board-specific functions start with `srtk` in their names.

The C functions can be used to program RTI-specific Simulink S-functions, or to implement your control models manually using C programs.

Examples

There are examples for some features that you will find after the installation of your dSPACE software in `<RCP_HIL_InstallationPath>\Demos\DS1202`. If there is a ZIP archive, you can open it as a project backup in ControlDesk to load and start the demo application on your real-time hardware.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
 DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
 WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
 CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.

Symbol	Description
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the icon in dSPACE Help. The PDF opens on the first page.

Basic Functions

Introduction

The board's RTLib provides functions for implementing the application framework, such as timer services and interrupt handling, and for accessing board components, such as LEDs and the buzzer.

Where to go from here

Information in this section

Data Types and Definitions.....	26
Initialization.....	28
Background Service.....	29
Time Interval Measurement.....	32
Time-Stamping.....	52
Timer A.....	65
Timer B.....	71
Timer D.....	77
Timer Interrupt Control.....	82
Interrupt Handling.....	93
Subinterrupt Handling.....	110
Message Handling.....	130
System Functions.....	153
Special Processor Functions.....	180
Conversion Functions.....	182
Standard Macros.....	186

Data Types and Definitions

Introduction

To present the elementary data types used with MicroLabBox.

Elementary Data Types

Data types

The `dstypes.h` file defines the overall processor-independent data types as follows:

<code>typedef signed char</code>	<code>Int8;</code>
<code>typedef unsigned char</code>	<code>UInt8;</code>
<code>typedef signed short</code>	<code>Int16;</code>
<code>typedef unsigned short</code>	<code>UInt16;</code>
<code>typedef signed int</code>	<code>Int32;</code>
<code>typedef unsigned int</code>	<code>UInt32;</code>
<code>typedef struct {UInt32 low; Int32 high;}</code>	<code>Int64;¹⁾</code>
<code>typedef struct {UInt32 low; UInt32 high;}</code>	<code>UInt64;¹⁾</code>
<code>typedef long long</code>	<code>Long64;</code>
<code>typedef unsigned long long</code>	<code>ULong64;</code>
<code>typedef float</code>	<code>Float32;</code>
<code>typedef double</code>	<code>Float64;</code>
<code>typedef double</code>	<code>dsfloat;</code>
<code>typedef Int8 *</code>	<code>Int8Ptr;</code>
<code>typedef UInt8 *</code>	<code>UInt8Ptr;</code>
<code>typedef Int16 *</code>	<code>Int16Ptr;</code>
<code>typedef UInt16 *</code>	<code>UInt16Ptr;</code>
<code>typedef Int32 *</code>	<code>Int32Ptr;</code>
<code>typedef UInt32 *</code>	<code>UInt32Ptr;</code>
<code>typedef Int64 *</code>	<code>Int64Ptr;</code>
<code>typedef UInt64 *</code>	<code>UInt64Ptr;</code>
<code>typedef Long64 *</code>	<code>Long64Ptr;</code>
<code>typedef ULong64 *</code>	<code>ULong64Ptr;</code>
<code>typedef Float32 *</code>	<code>Float32Ptr;</code>
<code>typedef Float64 *</code>	<code>Float64Ptr;</code>

¹⁾ The `Int64` and `UInt64` data types are deprecated and are provided only for backward-compatibility with older applications. It is recommended to use `Long64` and `ULong64` whenever 64 bit integer data types are required.

Include file `dstypes.h`

Initialization

init

Syntax	<code>init()</code>
Include file	<code>brtenv.h</code>
Purpose	To initialize all required hardware and software modules for MicroLabBox. It is recommended to use the <code>RTLIB_INIT</code> macro. For further information, refer to RTLIB_INIT on page 188.
<p>Note</p> <p>The initialization function <code>init</code> must be executed at the beginning of each application. It can only be invoked once. Further calls to this function are ignored.</p> <p>When you are using RTI, this function is called automatically in the simulation engine. Hence, you do not need to call <code>init</code> in S-functions. If you need to initialize single components that are not initialized by <code>init</code>, use the specific initialization functions that are described at the beginning of the function references.</p>	

Related topics	References
	RTLIB_INIT 188
	RTLIB_TERMINATE 189

Background Service

Where to go from here

Information in this section

[RTLIB_BACKGROUND_SERVICE](#)..... 29

To execute all relevant background functions with one call.

[rtlib_background_hook](#)..... 29

To register a specified hook function.

RTLIB_BACKGROUND_SERVICE

Syntax

`RTLIB_BACKGROUND_SERVICE()`

Include file

`SrtkStd.h`

Purpose

To call the essential functions in the model background loop.

Description

This macro executes all the required background services, for example, for the host communication. It must be continuously called in the background of your application, for example, within a `for` or a `while` construct. To constantly maintain its functionality, it must be called at least once per second.

Example

This is a code example for a background loop in an application program:

```
while(1)
{
    RTLIB_BACKGROUND_SERVICE();
}
```

rtlib_background_hook

Syntax

`int rtlib_background_hook(rtlib_bg_fcn_t *fcnptr)`

or

`RTLIB_REGISTER_BACKGROUND_HANDLER(rtlib_bg_fcn_t *fcnptr)`

Include file	SrtkStd.h						
Purpose	To register a function to be executed in the background loop.						
Description	You can register several functions by calling <code>rtlib_background_hook</code> subsequently.						
	<div style="border: 1px solid #ccc; padding: 5px;"><p>Note</p><ul style="list-style-type: none">▪ The specified function must be of type <code>rtlib_bg_fcn_t</code>, which defines a function with no arguments and no return value.▪ The background loop waits for the execution of the specified hook functions. Ensure that the hook functions do not completely block the background service.</div>						
Parameters	fcnptr Specifies the pointer to the background function.						
Return value	This function returns the following values:						
	<table border="1" style="width: 100%; border-collapse: collapse;"><thead><tr><th style="text-align: left; padding: 2px;">Return Value</th><th style="text-align: left; padding: 2px;">Meaning</th></tr></thead><tbody><tr><td style="text-align: left; padding: 2px;">0</td><td style="text-align: left; padding: 2px;">The background function has been registered successfully.</td></tr><tr><td style="text-align: left; padding: 2px;">-1</td><td style="text-align: left; padding: 2px;">An error occurred while registering the background function.</td></tr></tbody></table>	Return Value	Meaning	0	The background function has been registered successfully.	-1	An error occurred while registering the background function.
Return Value	Meaning						
0	The background function has been registered successfully.						
-1	An error occurred while registering the background function.						
Example	This example shows how to implement a simple hook function within the background loop. The variable <code>bg_count</code> counts the number of executed background loops. <pre style="background-color: #f0f0f0; padding: 10px;">int bg_count=0; void bg_fcn() { bg_count++; } void main(void) { int result; init(); /* setup foreground, for e.g. a timer isr */ ... result = rtlib_background_hook(bg_fcn); ... }</pre>						

```
/* background loop */
while(1)
{
    /* call the background functions */
    RTLIB_BACKGROUND_SERVICE();
}
```

Related topics**References**

RTLIB_BACKGROUND_SERVICE	29
--	----

Time Interval Measurement

Introduction

Functions for measuring time intervals are used for profiling application code (execution time measurement) or for implementing time delays. The time is derived from the built-in PowerPC time base, which has a resolution of 25 MHz.

Tip

Here you find the descriptions of platform-specific functions and generic `RTLIB_TIC_XXX` macros. It is recommended to use the generic macros.

Where to go from here

Information in this section

Data Types for Time Measurement	33
Example of Using Time Measurement Functions	34
<code>srtk_tic_continue</code>	34
To resume time measurement after it was paused.	
<code>srtk_tic_count</code>	35
To read the current counter value of the time base.	
<code>srtk_tic_delay</code>	36
To perform the specified time delay.	
<code>srtk_tic_diff</code>	37
To calculate the difference between two time base counter values.	
<code>srtk_tic_elapsed</code>	38
To calculate the difference between a previous time base counter value and the current time base value.	
<code>srtk_tic_halt</code>	39
To pause time measurement.	
<code>srtk_tic_read</code>	40
To read the time period since time measurement was started, minus the breaks.	
<code>srtk_tic_start</code>	41
To start a time measurement.	
<code>srtk_tic_total_read</code>	41
To read the complete time period since the time measurement was started, including all breaks.	
<code>srtk_timebase_fltread</code>	42
To read the 64-bit value of the time base register and convert the result to a 64-bit float value (seconds).	

srtk_timebase_low_read	42
To read the lower 32 bits of the time base register.	
srtk_timebase_read	43
To read the 64 bits of the time base register.	
RTLIB_TIC_CONTINUE	44
To resume time measurement after it was paused.	
RTLIB_TIC_COUNT	44
To read the current counter value of the time base.	
RTLIB_TIC_DELAY	45
To perform the specified time delay.	
RTLIB_TIC_DIFF	46
To calculate the difference between two time base counter values.	
RTLIB_TIC_ELAPSED	47
To calculate the difference between a previous time base counter value and the current time base value.	
RTLIB_TIC_HALT	48
To pause time measurement.	
RTLIB_TIC_READ	49
To read the time period since time measurement was started minus the breaks made.	
RTLIB_TIC_READ_TOTAL	50
To read the complete time period since the time measurement was started, including all breaks made.	
RTLIB_TIC_START	50
To start a time measurement.	

Data Types for Time Measurement

Introduction

There is one specific data type used by the `srtk_tic_count`, `srtk_tic_elapsed`, `srtk_tic_diff` functions and their related macros.

rtlib_tic_t

This data type is used to specify the time base counter values. It is defined as `ULong64` data type.

Example of Using Time Measurement Functions

Example

The following example shows the source code to measure the execution time of certain actions. Three actions are specified in the program, but only action 1 and action 3 are measured using the board-specific function names:

```
srtk_tic_start(); /* starts time measurement */
...
time = srtk_tic_read();
... action 1 ...
srtk_tic_halt(); /* start of the break */
... action 2 ...
srtk_tic_continue(); /* end of the break */
... action 3 ...
time = srtk_tic_read() - time;
/* second read and calculation of the action 1 and 3 period */
```

To measure the execution time of action 1 and action 3 using the standard macros:

```
RTLIB_TIC_START(); /* starts time measurement */
...
time = RTLIB_TIC_READ();
... action 1 ...
RTLIB_TIC_HALT(); /* start of the break */
... action 2 ...
RTLIB_TIC_CONTINUE(); /* end of the break */
... action 3 ...
time = RTLIB_TIC_READ() - time;
/* second read and calculation of the action 1 and 3 period */
```

srtk_tic_continue

Syntax

```
srtk_tic_continue()
```

Include file

SrtkTick.h

Purpose

To resume time measurement after it was paused by `srtk_tic_halt`.

Description

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 54.

Return value

None

Related topics**Examples**

Example of Using Time Measurement Functions..... 34

References

RTLIB_TIC_CONTINUE..... 44
srtk_tic_halt..... 39

srtk_tic_count

Syntax

```
rtlib_tic_t srtk_tic_count(void)
```

Include file**SrtkTick.h**

Purpose

To read the current counter value of the time base.

Description

Use **srtk_tic_count** in conjunction with **srtk_tic_elapsed** or **srtk_tic_diff** to perform execution time measurement in recursive functions.

Parameters

None

Return value

This function returns the current counter value of the time base as **rtlib_tic_t** data type.

Example

The following example shows how to calculate the time difference between two time base counter values.

```
void main(void)
{
    rtplib_tic_t timer_count1 = 0,
    rtplib_tic_t timer_count2 = 0;
    dsfloat exec_time = 0;

    init();

    timer_count1 = srtk_tic_count();
    ...
    timer_count2 = srtk_tic_count();
    exec_time = srtk_tic_diff(timer_count1, timer_count2);
    ...
}
```

Related topics**References**

RTLIB_TIC_COUNT.....	44
srtk_tic_diff.....	37
srtk_tic_elapsed.....	38

srtk_tic_delay

Syntax

```
srtk_tic_delay(Float64 duration)
```

Include file

SrtkTick.h

Purpose

To perform the specified time delay.

Parameters

duration Specifies the time delay in seconds. If you specify a duration that exceeds the maximum range of the timer, the function never stops.

Return value

None

Related topics**References**

RTLIB_TIC_DELAY.....	.45
srtk_tic_continue.....	.34
srtk_tic_start.....	.41

srtk_tic_diff

Syntax

```
dsfloat srtk_tic_diff(  
    rtlib_tic_t tmr_cnt1,  
    rtlib_tic_t tmr_cnt2)
```

Include file**SrtkTick.h**

Purpose

To calculate the difference between two time base counter values.

Description

Use **srtk_tic_diff** in conjunction with **srtk_tic_count** or **srtk_tic_elapsed** to perform execution time measurement in recursive functions.

Parameters

tmr_cnt1 Specifies the first time base counter value.

tmr_cnt2 Specifies the second time base counter value.

Return value

This function returns the time difference in seconds.

Example

The following example shows how to calculate the time difference between two time base counter values.

```
void main(void)
{
    rtlib_tic_t timer_count1 = 0, timer_count2 = 0;
    dsfloat exec_time = 0;

    init();

    timer_count1 = srtk_tic_count();
    ...
    timer_count2 = srtk_tic_count();
    exec_time = srtk_tic_diff(timer_count1, timer_count2);
    ...
}
```

Related topics**References**

RTLIB_TIC_DIFF	46
srtk_tic_count	35
srtk_tic_elapsed	38

srtk_tic_elapsed

Syntax

```
dsfloat srtk_tic_elapsed(rtlib_tic_t tmr_cnt)
```

Include file

SrtkTick.h

Purpose

To calculate the difference between a previous time base counter value specified by **tmr_cnt** and the current time base value in seconds.

Description

Use **srtk_tic_elapsed** in conjunction with **srtk_tic_count** or **srtk_tic_diff** to perform execution time measurement in recursive functions.

Parameters

tmr_cnt Specifies the previous counter value of the time base.

Return value

This function returns the elapsed time in seconds.

Example

The following example shows how to calculate the time difference between a previous time base counter value and the current time base value.

```
void main(void)
{
    rtplib_tic_t timer_count;
    dsfloat exec_time = 0;

    init();

    timer_count = srtk_tic_count();
    ...
    exec_time = srtk_tic_elapsed(timer_count);
    ...
}
```

Related topics**References**

RTLIB_TIC_ELAPSED.....	47
srtk_tic_count.....	35
srtk_tic_diff.....	37

srtk_tic_halt

Syntax`srtk_tic_halt()`**Include file**

SrtkTick.h

Purpose

To pause time measurement.

DescriptionThe break lasts until measurement is resumed by `srtk_tic_continue`.

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 54.

Return value

None

Related topics**Examples**

Example of Using Time Measurement Functions.....34

ReferencesRTLIB_TIC_HALT.....48
srtk_tic_continue.....34

srtk_tic_read

Syntax`Float64 srtk_tic_read()`**Include file**

SrtkTick.h

Purpose

To read the time period since time measurement was started by `srtk_tic_start`, minus the breaks made from `srtk_tic_halt` to `srtk_tic_continue`.

Description

Use `srtk_tic_total_read` to read the complete time period including the breaks.

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 54.

Return value

This function returns the time duration in seconds.

Related topics**Examples**

Example of Using Time Measurement Functions.....34

ReferencesRTLIB_TIC_READ.....49
srtk_tic_continue.....34
srtk_tic_halt.....39
srtk_tic_start.....41
srtk_tic_total_read.....41

srtk_tic_start

Syntax	<code>srtk_tic_start()</code>
Include file	<code>SrtkTick.h</code>
Purpose	To start a time measurement.
Description	This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to Time-Stamping Functions on page 54.
Return value	None
Related topics	<p>Examples</p> <p>Example of Using Time Measurement Functions..... 34</p> <p>References</p> <p>RTLIB_TIC_START..... 50</p>

srtk_tic_total_read

Syntax	<code>Float64 srtk_tic_total_read()</code>
Include file	<code>SrtkTick.h</code>
Purpose	To read the complete time period since the time measurement was started by <code>srtk_tic_start</code> , including all breaks made from <code>srtk_tic_halt</code> to <code>srtk_tic_continue</code> .
Description	<p>Use <code>srtk_tic_read</code> to read the time period minus the breaks made.</p> <p>This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to Time-Stamping Functions on page 54.</p>

Return value	This function returns the time duration in seconds.
---------------------	---

Related topics	References
-----------------------	------------

RTLIB_TIC_READ_TOTAL.....	50
srtk_tic_continue.....	34
srtk_tic_halt.....	39
srtk_tic_read.....	40

srtk_timebase_fltread

Syntax	<code>Float64 srtk_timebase_fltread(void)</code>
---------------	--

Include file	<code>SrtkTmr.h</code>
---------------------	------------------------

Purpose	To read the 64-bit value of the time base register and convert the result to a 64-bit float value (seconds).
----------------	--

Return value	This function returns the current value of the time base register in seconds.
---------------------	---

Related topics	References
-----------------------	------------

srtk_timebase_low_read.....	42
srtk_timebase_read.....	43

srtk_timebase_low_read

Syntax	<code>UInt32 srtk_timebase_low_read(void)</code>
---------------	--

Include file	<code>SrtkTmr.h</code>
---------------------	------------------------

Purpose	To read the lower 32 bits of the time base register.
----------------	--

Description	Use <code>srtk_timebase_read</code> to read the complete time base register.
	<div style="border: 1px solid #ccc; padding: 5px;"><p>Note</p><p>This function is provided for downward compatibility and should not be used on MicroLabBox (DS1202)</p><p>The time base of MicroLabBox has a resolution of 25 MHz. The lower time base register (TBRL) will wrap to zero after nearly three minutes. This causes problems if the TBRL is used for interval measurements or delays greater than three minutes.</p></div>
Return value	This function returns the lower 32 bits of the current time base register.
Related topics	References

[srtk_timebase_fltread](#).....42
[srtk_timebase_read](#).....43

srtk_timebase_read

Syntax	<code>ULong64 srtk_timebase_read(void)</code>
Include file	<code>SrtkTmr.h</code>
Purpose	To read the 64 bits of the time base register.
Return value	This function returns the current value of the time base register.
Related topics	References

[srtk_timebase_fltread](#).....42
[srtk_timebase_low_read](#).....42

RTLIB_TIC_CONTINUE

Syntax	<code>RTLIB_TIC_CONTINUE()</code>
Include file	<code>SrtkStd.h</code>
Purpose	To resume time measurement after it was paused by RTLIB_TIC_HALT . This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to Time-Stamping Functions on page 54.
Return value	None
Related topics	<p>Examples</p> <p>Example of Using Time Measurement Functions..... 34</p> <p>References</p> <p>RTLIB_TIC_HALT..... 48 srtk_tic_continue..... 34</p>

RTLIB_TIC_COUNT

Syntax	<code>rtlib_tic_t RTLIB_TIC_COUNT(void)</code>
Include file	<code>SrtkStd.h</code>
Purpose	To read the current counter value of the time base.
Description	Use RTLIB_TIC_COUNT() in conjunction with RTLIB_TIC_ELAPSED or RTLIB_TIC_DIFF to perform execution time measurement in recursive functions.
Parameters	None

Return value	This function returns the current counter value of the time base as <code>rtlib_tic_t</code> data type.
---------------------	---

Example	The following example shows how to calculate the time difference between two time base counter values.
----------------	--

```
void main(void)
{
    rtlib_tic_t timer_count1 = 0,
    rtlib_tic_t timer_count2 = 0;
    dsfloat exec_time = 0;

    init();

    timer_count1 = RTLIB_TIC_COUNT();
    ...
    timer_count2 = RTLIB_TIC_COUNT();
    exec_time = RTLIB_TIC_DIFF(timer_count1, timer_count2);
}
```

Related topics	References
	RTLIB_TIC_DIFF.....46
	RTLIB_TIC_ELAPSED.....47
	srtk_tic_count.....35

RTLIB_TIC_DELAY

Syntax	<code>RTLIB_TIC_DELAY(Float64 duration)</code>
Include file	<code>SrtkStd.h</code>
Purpose	To perform the specified time delay.
Parameters	duration Specifies the time delay in seconds. If you specify a duration that exceeds the maximum range of the timer, the function never stops.
Return value	None

Related topics	References						
	<table><tr><td>RTLIB_TIC_CONTINUE.....</td><td>44</td></tr><tr><td>RTLIB_TIC_START.....</td><td>50</td></tr><tr><td>srtk_tic_delay.....</td><td>36</td></tr></table>	RTLIB_TIC_CONTINUE.....	44	RTLIB_TIC_START.....	50	srtk_tic_delay.....	36
RTLIB_TIC_CONTINUE.....	44						
RTLIB_TIC_START.....	50						
srtk_tic_delay.....	36						

RTLIB_TIC_DIFF

Syntax	<pre>dsfloat RTLIB_TIC_DIFF(rtlib_tic_t tmr_cnt1, rtlib_tic_t tmr_cnt2)</pre>				
Include file	SrtkStd.h				
Purpose	To calculate the difference between two time base counter values.				
Description	Use RTLIB_TIC_DIFF in conjunction with RTLIB_TIC_COUNT or RTLIB_TIC_ELAPSED to perform execution time measurement in recursive functions.				
Parameters	<table><tr><td>tmr_cnt1</td><td>Specifies the first time base counter value.</td></tr><tr><td>tmr_cnt2</td><td>Specifies the second time base counter value.</td></tr></table>	tmr_cnt1	Specifies the first time base counter value.	tmr_cnt2	Specifies the second time base counter value.
tmr_cnt1	Specifies the first time base counter value.				
tmr_cnt2	Specifies the second time base counter value.				
Return value	This function returns the time difference in seconds.				

Example

The following example shows how to calculate the time difference between two time base counter values.

```
void main(void)
{
    rtplib_tic_t timer_count1 = 0, timer_count2 = 0;
    dsfloat exec_time = 0;

    init();

    timer_count1 = RTLlib_TIC_COUNT();
    ...
    timer_count2 = RTLlib_TIC_COUNT();
    exec_time = RTLlib_TIC_DIFF(timer_count1, timer_count2);
    ...
}
```

Related topics**References**

RTLIB_TIC_COUNT	44
RTLIB_TIC_ELAPSED	47
srtk_tic_diff	37

RTLIB_TIC_ELAPSED

Syntax

```
dsfloat RTLIB_TIC_ELAPSED(rtlLib_tic_t tmr_cnt)
```

Include file

SrtkStd.h

Purpose

To calculate the difference between a previous time base counter value specified by `tmr_cnt` and the current time base value in seconds using a generic macro.

Description

Use `RTLIB_TIC_ELAPSED` in conjunction with `RTLIB_TIC_COUNT` or `RTLIB_TIC_DIFF` to perform execution time measurement in recursive functions.

Parameters

`tmr_cnt` Specifies the previous counter value of the time base.

Return value

This function returns the elapsed time in seconds.

Example

The following example shows how to calculate the time difference between a previous time base counter value and the current time base value.

```
void main(void)
{
    rtplib_tic_t timer_count;
    dsfloat exec_time = 0;

    init();

    timer_count = RTLlib_TIC_COUNT();
    ...
    exec_time = RTLlib_TIC_ELAPSED(timer_count);
    ...
}
```

Related topics**References**

[srtk_tic_elapsed](#).....38

RTLIB_TIC_HALT

Syntax

`RTLIB_TIC_HALT()`

Include file

`SrtkStd.h`

Purpose

To pause time measurement.

Description

The break lasts until measurement is resumed by `RTLIB_TIC_CONTINUE`.

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 54.

Return value

None

Related topics**Examples**

Example of Using Time Measurement Functions.....34

References

RTLIB_TIC_CONTINUE.....44
srtk_tic_halt.....39

RTLIB_TIC_READ

Syntax

`RTLIB_TIC_READ()`

Include file

`SrtkStd.h`

Purpose

To read the time period since time measurement was started by `RTLIB_TIC_START`, minus the breaks made from `RTLIB_TIC_HALT` to `RTLIB_TIC_CONTINUE`.

Description

Use `RTLIB_TIC_READ_TOTAL` to read the complete time period including the breaks made.

This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to [Time-Stamping Functions](#) on page 54.

Return value

This function returns the time duration in seconds.

Related topics**Examples**

Example of Using Time Measurement Functions.....34

References

RTLIB_TIC_CONTINUE.....44
RTLIB_TIC_HALT.....48
RTLIB_TIC_START.....50
srtk_tic_read.....40

RTLIB_TIC_READ_TOTAL

Syntax	<code>RTLIB_TIC_READ_TOTAL()</code>										
Include file	<code>SrtkStd.h</code>										
Purpose	To read the complete time period since the time measurement was started by <code>RTLIB_TIC_START</code> , including all breaks made from <code>RTLIB_TIC_HALT</code> to <code>RTLIB_TIC_CONTINUE</code> .										
Description	<p>Use <code>RTLIB_TIC_READ</code> to read the time period minus the breaks made.</p> <p>This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to Time-Stamping Functions on page 54.</p>										
Return value	This function returns the time duration in seconds.										
Related topics	References <table><tr><td><code>RTLIB_TIC_CONTINUE</code>.....</td><td>44</td></tr><tr><td><code>RTLIB_TIC_HALT</code>.....</td><td>48</td></tr><tr><td><code>RTLIB_TIC_READ</code>.....</td><td>49</td></tr><tr><td><code>RTLIB_TIC_START</code>.....</td><td>50</td></tr><tr><td><code>srtk_tic_total_read</code>.....</td><td>41</td></tr></table>	<code>RTLIB_TIC_CONTINUE</code>	44	<code>RTLIB_TIC_HALT</code>	48	<code>RTLIB_TIC_READ</code>	49	<code>RTLIB_TIC_START</code>	50	<code>srtk_tic_total_read</code>	41
<code>RTLIB_TIC_CONTINUE</code>	44										
<code>RTLIB_TIC_HALT</code>	48										
<code>RTLIB_TIC_READ</code>	49										
<code>RTLIB_TIC_START</code>	50										
<code>srtk_tic_total_read</code>	41										

RTLIB_TIC_START

Syntax	<code>RTLIB_TIC_START()</code>
Include file	<code>SrtkStd.h</code>
Purpose	To start a time measurement.
Description	This function is not reentrant. It is recommended to use the time-stamping functions instead, refer to Time-Stamping Functions on page 54.

Return value None

Related topics Examples

Example of Using Time Measurement Functions..... 34

References

srtk_tic_start..... 41

Time-Stamping

Introduction	The time-stamping module is used to take absolute time stamps from a highly accurate, absolute time base.
---------------------	---

Where to go from here	Information in this section
	General Information on Time-Stamping 52
	Data Types and Global Variables for Time-Stamping 54
	Time-Stamping Functions 54

General Information on Time-Stamping

Introduction	Gives you information on basic principles and implementation details of the time-stamping feature.
---------------------	--

Where to go from here	Information in this section
	Basic Principles of Time-Stamping 52
	Principles of an Absolute Time in Single-Processor and Multiprocessor Systems 53

Basic Principles of Time-Stamping

Introduction	The Time-Stamping module is used to take absolute time stamps from a highly accurate, absolute time base. The time base fulfills the following requirements:
---------------------	--

Time stamp accuracy The exact resolution depends on the mode of the Time-Stamping module. See [Modes of the Time-Stamping module](#) on page 53 for the exact resolution.

Time stamp range The time base has a range of 64 bit. Combined with a resolution down to 40 ns, this is enough to measure highly accurate absolute times up to several years.

Principles of an Absolute Time in Single-Processor and Multiprocessor Systems

Introduction

The Time-Stamping module is the fundamental time base for real-time simulations. It provides sufficiently accurate samples of the independent variable time. Therefore, if data and events have been recorded together with the associated time stamps, it is possible to reconstruct their temporal order.

Note

The same information applies to multiprocessor applications running on a multicore system.

Synchronization of local clocks

Each processor has its own local time base (local clock). Due to manufacturing tolerances, which lead to clock drifts, the local clocks in a multiprocessor system have to be synchronized periodically. To keep the communication effort low, synchronization does not take place at every tick of the local clocks (microtick), but at a selected tick of a timing master. This selected tick is called macrotick.

In single-processor systems, no synchronization is required. In a single-processor system, one macrotick equals 2^{32} microticks with the microtick running at the speed of the CPU's time base. This means that macrotick and microtick are actually stored in a 64-bit value. The data type of the time stamp structure contains one counter for the microtick and one for the macrotick. Because of this, the time stamp structure meets the requirements of both single-processor and multiprocessor systems.

When a macrotick occurs, the number of microticks is set to zero and the number of macroticks is increased by 1 at each processor. Starting from this point in time, the absolute time t_{abs} is calculated as follows:

$$t_{abs} = MAT \cdot P_{MAT} + MIT \cdot P_{MIT}$$

In this equation, "MAT" denotes the number of macroticks, which is incremented in the entire system, whereas "MIT" is the number of microticks. " P_{MAT} " is the macrotick period, which is a system-wide constant. " P_{MIT} " denotes the microtick period that can differ from clock to clock.

Modes of the Time-Stamping module

The Time-Stamping module can operate in three different modes:

single mode This is the mode for single-processor systems (single-core applications).

The microtick (the tick of the local clock) is directly taken from the CPU-internal time-base register. It has a resolution of 40 ns (25 MHz) on MicroLabBox.

multi-master mode This is the mode of the timing master in a multiprocessor system (multicore application).

The microtick is generated by the synchronous time base unit (STBU), and driven by the bus clock of MicroLabBox, scaled by 2. For example, at a MicroLabBox with 100 MHz bus clock, the resolution of the Microtick Counter is 20 ns.

When the Microtick Counter reaches the macrotick period, a system-wide macrotick is generated.

multi-slave mode This is the mode of all other processors in a multiprocessor system (multicore application).

As on the master processor, the microtick is generated by the STBU. Processors in *Slave mode* receive their macrotick from the timing master.

Data Types and Global Variables for Time-Stamping

Introduction

Gives you basic information on data types and global variables used for time-stamping.

Data Types Used for Time-Stamping

Data types

The following data types are defined for time-stamping:

Data Type	Syntax
ts_timestamp_type	<pre>typedef struct { UInt32 mat; /* 32 bit macrotick counter value */ UInt32 mit; /* 32 bit microtick counter value */ }ts_timestamp_type;</pre>
ts_timestamp_ptr_type	<code>typedef ts_timestamp_type * ts_timestamp_ptr_type</code>

Time-Stamping Functions

Introduction

Gives you information on the C functions available for the time-stamping feature.

Where to go from here**Information in this section**

ts_init	55
To initialize the Time-Stamping module.	
ts_mat_period_get	57
To get the time for one macrotick period.	
ts_mit_period_get	57
To get the time for one microtick period.	
ts_reset	58
To set the absolute time to 0.	
ts_time_read	58
To read the absolute time in seconds.	
ts_timestamp_read	59
To read the absolute time and return it as time stamp structure.	
ts_timestamp_compare	60
To compare two time stamps.	
ts_timestamp_interval	61
To return the interval between two time stamps.	
ts_time_offset	61
To calculate the difference between two time stamps and add this difference to the reference time.	
ts_timestamp_offset	62
To calculate the difference between two time stamps and add this difference to the reference time stamp.	
ts_time_calculate	63
To convert a time stamp structure to a time value in seconds.	
ts_timestamp_calculate	64
To convert a time value in seconds to a time stamp structure.	

[ts_init](#)

Syntax

```
int ts_init(
    int mode,
    float mat_period)
```

Include file

dsts.h

Purpose	To initialize the Time-Stamping module and the hardware, and to reset the Microtick and the Macrotick Counter.								
Description	<ul style="list-style-type: none"> ▪ The function <code>ts_init</code> is called automatically during board initialization. The Time-Stamping module is set to the <code>TS_MODE_SINGLE</code> mode. ▪ The function <code>ts_init</code> is also called automatically by the multiprocessor initialization, which sets the Time-Stamping module to the <code>TS_MODE_MULTI_MASTER</code> mode at the processor core with ID 0 and to the <code>TS_MODE_MULTI_SLAVE</code> mode at the other core. ▪ When the Time-Stamping module is initialized with <code>TS_MODE_MULTI_MASTER</code> or <code>TS_MODE_MULTI_SLAVE</code>, the Synchronous Time Base Unit (STBU) is stopped. It can be started explicitly by calling <code>ts_reset</code>. 								
Parameters	<p>mode Specifies the mode of the Time-Stamping module; the following symbols are predefined:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Predefined Symbol</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td><code>TS_MODE_SINGLE</code></td> <td>single mode</td> </tr> <tr> <td><code>TS_MODE_MULTI_MASTER</code></td> <td>multi-master mode</td> </tr> <tr> <td><code>TS_MODE_MULTI_SLAVE</code></td> <td>multi-slave mode</td> </tr> </tbody> </table> <p>mat_period Specifies the time in seconds of one macrotick period. In single-processor systems, this argument is ignored (can be 0.0).</p>	Predefined Symbol	Meaning	<code>TS_MODE_SINGLE</code>	single mode	<code>TS_MODE_MULTI_MASTER</code>	multi-master mode	<code>TS_MODE_MULTI_SLAVE</code>	multi-slave mode
Predefined Symbol	Meaning								
<code>TS_MODE_SINGLE</code>	single mode								
<code>TS_MODE_MULTI_MASTER</code>	multi-master mode								
<code>TS_MODE_MULTI_SLAVE</code>	multi-slave mode								
Return value	This function returns the error code; the following symbols are predefined:								

Related topics

Basics

Basic Principles of Time-Stamping.....	52
--	----

References

ts_reset.....	58
-------------------------------	----

ts_mat_period_get

Syntax

```
dsfloat ts_mat_period_get()
```

Include file

dsts.h

Purpose

To get the time for one macrotick period.

Return value

Returns the macrotick period in seconds.

Related topics

Basics

Basic Principles of Time-Stamping..... 52

References

ts_init..... 55
ts_mit_period_get..... 57

ts_mit_period_get

Syntax

```
dsfloat ts_mit_period_get()
```

Include file

dsts.h

Purpose

To get the time for one microtick period.

Description

The microtick depends on the frequency of the Time Base Counter.

Return value

Returns the microtick period in seconds.

Related topics

Basics

[Basic Principles of Time-Stamping](#).....52

References

[ts_init](#).....55

[ts_mat_period_get](#).....57

ts_reset

Syntax

`void ts_reset()`

Include file

`dsts.h`

Purpose

To reset the Time-Stamping module to the absolute time 0.

Return value

None

Related topics

Basics

[Basic Principles of Time-Stamping](#).....52

References

[ts_init](#).....55

ts_time_read

Syntax

`double ts_time_read()`

Include file

`dsts.h`

Purpose	To read the absolute time in seconds.
Return value	This function returns the absolute time in seconds since the initialization <code>ts_init</code> or the last reset <code>ts_reset</code> .
Related topics	<p>Basics</p> <div style="background-color: #f0f0f0; padding: 5px;"> Basic Principles of Time-Stamping..... 52 </div> <p>References</p> <div style="background-color: #f0f0f0; padding: 5px;"> ts_timestamp_read..... 59 </div>

ts_timestamp_read

Syntax	<code>void ts_timestamp_read(ts_timestamp_ptr_type ts)</code>
Include file	<code>dsts.h</code>
Purpose	To read the absolute time and return it as time stamp structure.
Result	The absolute time is read and is written to the time stamp structure <code>ts</code> points to.
Parameters	ts Specifies the pointer to a time stamp structure for the read value.
Return value	None
Related topics	<p>Basics</p> <div style="background-color: #f0f0f0; padding: 5px;"> Basic Principles of Time-Stamping..... 52 </div> <p>References</p> <div style="background-color: #f0f0f0; padding: 5px;"> ts_time_read..... 58 </div>

ts_timestamp_compare

Syntax

```
int ts_timestamp_compare(
    ts_timestamp_ptr_type ts1,
    ts_timestamp_ptr_type ts2,
    int operation)
```

Include file

dsts.h

Purpose

To compare two time stamps.

Parameters

ts1 Specifies the pointer to the first time stamp structure.

ts2 Specifies the pointer to the second time stamp structure.

operation Specifies the kind of operation; the following symbols are predefined:

Predefined Symbol	Meaning
TS_COMPARE_LT	less than
TS_COMPARE_LE	less than or equal to
TS_COMPARE_EQ	equal
TS_COMPARE_GE	greater than or equal to
TS_COMPARE_GT	greater than

Return value

This function returns the operation result; the following symbols are predefined:

Value	Meaning
= 0	Result is false
!= 0	Result is true

Related topics

Basics

[Basic Principles of Time-Stamping.....](#) 52

References

[ts_timestamp_interval.....](#) 61

ts_timestamp_interval

Syntax

```
double ts_timestamp_interval(  
    ts_timestamp_ptr_type ts1,  
    ts_timestamp_ptr_type ts2)
```

Include file**dsts.h**

Purpose

To calculate the interval in seconds between time stamps 1 and 2.

Parameters**ts1** Specifies the pointer to the first time stamp structure.**ts2** Specifies the pointer to the second time stamp structure.

Return value

This function returns the interval between time stamps 1 and 2 in seconds.

Related topics**Basics**[Basic Principles of Time-Stamping.....](#) 52**References**[ts_timestamp_compare.....](#) 60

ts_time_offset

Syntax

```
void ts_time_offset(  
    double reference_time,  
    ts_timestamp_ptr_type ts1,  
    ts_timestamp_ptr_type ts2,  
    ts_timestamp_ptr_type ts_ta)
```

Include file**dsts.h**

Purpose

To calculate the time offset.

Result	The interval between time stamps 1 and 2 is calculated and the difference between the time stamps is added to the reference time. The absolute time is returned as a time stamp.
---------------	--

Parameters	reference_time Specifies the reference time in seconds. ts1 Specifies the pointer to the first time stamp structure. ts2 Specifies the pointer to the second time stamp structure. ts_ta Specifies the pointer to the time stamp structure for the calculated value.
-------------------	---

Return value	None
---------------------	------

Related topics	Basics Basic Principles of Time-Stamping.....52
	References ts_timestamp_offset.....62

ts_timestamp_offset

Syntax	<pre>void ts_timestamp_offset(ts_timestamp_ptr_type ts_reference, ts_timestamp_ptr_type ts1, ts_timestamp_ptr_type ts2, ts_timestamp_ptr_type ts_ta)</pre>
---------------	---

Include file	dsts.h
---------------------	--------

Purpose	To calculate the time offset.
----------------	-------------------------------

Result	The interval between time stamps 1 and 2 is calculated and the difference between the time stamps is added to the reference time stamp. The absolute time is returned as a time stamp.
---------------	--

Parameters	ts_reference Specifies the pointer to the time stamp structure holding the reference time. ts1 Specifies the pointer to the first time stamp structure. ts2 Specifies the pointer to the second time stamp structure. ts_ta Specifies the pointer to the time stamp structure holding the absolute time in seconds.
Return value	None
Related topics	Basics Basic Principles of Time-Stamping.....52 References ts_time_offset.....61

ts_time_calculate

Syntax `double ts_time_calculate(ts_timestamp_ptr_type ts)`

Include file `dsts.h`

Purpose To convert a time stamp structure to a time value in seconds.

Parameters **ts** Specifies the pointer to a time stamp structure.

Return value This function returns the time corresponding to the time stamp.

Related topics

Basics

[Basic Principles of Time-Stamping](#).....52

References

[ts_timestamp_offset](#).....62

ts_timestamp_calculate

Syntax

```
void ts_time_calculate(  
    double time,  
    ts_timestamp_ptr_type ts)
```

Include file

dsts.h

Purpose

To convert a time value in seconds to a time stamp structure.

Parameters

time Specifies the time in seconds.

ts Specifies the pointer to a time stamp structure for the calculated value.

Return value

None

Related topics

Basics

[Basic Principles of Time-Stamping](#).....52

References

[ts_time_calculate](#).....63

Timer A

Introduction

Timer A is a down counter generating an interrupt whenever it reaches zero. The period value is then reloaded automatically. Timer A is also used by the `RTLIB_SRT_PERIOD` standard macro as the default sampling rate timer.

For further information on Timer A, refer to [Timer Services \(MicroLabBox Features\)](#).

Where to go from here

Information in this section

Example of Using Timer A Functions	65
Gives you an example how to use Timer A.	
RTLIB_SRT_PERIOD	66
To set a new period of Timer A and restart it immediately.	
srtk_timerA_period_set	67
To set the period of Timer A.	
srtk_timerA_period_reload	67
To set a new period of Timer A and restart it immediately.	
srtk_timerA_read	68
To read the current value of Timer A.	
srtk_timerA_start	69
To start Timer A.	
srtk_timerA_stop	69
To stop Timer A.	

Information in other sections

For information on handling Timer A interrupts	
Timer Interrupt Control	82
Interrupt Handling	93

Example of Using Timer A Functions

Example

The following example demonstrates how to use Timer A functions.

```
#include <Brtenv.h>
#define DT 1.0e-4           /* 100 µs simulation step size */
```

```

/* ++ variables for host PC ++++++ */
Float64 exec_time, timeA;      /* execution time */
void ad_routine(void)
{
    ts_timestamp_type ts;
    Float64 old_timeA;
    RTLIB_SRT_ISR_BEGIN();           /* overrun check TimerA */
    srtk_timerA_read(&old_timeA);

    ts_timestamp_read(&ts);
    DsDaq_Service(0, 0, 1,
                  (DsDaqSTimestampStruct *)&ts); /* data acquisition service */
    /* +++ do something +++ */
    srtk_timerA_read(&timeA);
    exec_time = old_timeA - timeA; /* exec time with Timer A */
    RTLIB_SRT_ISR_END();           /* overrun check TimerA */
}
void main(void)
{
    /* init processor board */
    init();
    /* set period of timerA */
    timeA = DT;
    srtk_timerA_period_set(timeA);
    /* periodic event in ISR */
    RTLIB_SRT_START(timeA, ad_routine);
    /* Background tasks */
    while(1)
    {
        RTLIB_BACKGROUND_SERVICE(); /* host PC service */
    }
}

```

RTLIB_SRT_PERIOD

Syntax	<code>RTLIB_SRT_PERIOD(Float64 time)</code>
Include file	<code>SrtkStd.h</code>
Purpose	To set a new period of Timer A and restart it immediately.
Description	The new value is loaded immediately: Timer A is stopped, the new value is set, and Timer A is started again.
Parameters	time Specifies the period in seconds.

Return value None

Related topics References

[srtk_timerA_period_set](#).....67

srtk_timerA_period_set

Syntax `void srtk_timerA_period_set(Float64 time)`

Include file SrtkTmr.h

Purpose To set the period of Timer A.

Description If the timer is not running, the new value is loaded immediately. If the timer is running, the new value is loaded the next time the timer reaches zero.

Parameters **time** Specifies the period in seconds.

Return value None

Related topics Examples

[Example of Using Timer A Functions](#).....65

References

[srtk_timerA_period_reload](#).....67

srtk_timerA_period_reload

Syntax `void srtk_timerA_period_reload(Float64 time)`

Include file	SrtkTmr.h
Purpose	To set a new period of Timer A and restart it immediately.
Description	The new value is loaded immediately: Timer A is stopped, the new value is set, and Timer A is started again.
Parameters	time Specifies the period in seconds.
Return value	None
Related topics	References
	RTLIB_SRT_PERIOD..... 66 srtk_timerA_period_set..... 67

srtk_timerA_read

Syntax	<code>void srtk_timerA_read(Float64 *time)</code>
Include file	SrtkTmr.h
Purpose	To read the current value of Timer A.
Parameters	time Specifies the pointer to the current value of Timer A. The value is stated in seconds.
Return value	None

srtk_timerA_start

Syntax

```
void srtk_timerA_start(void)
```

Include file

SrtkTmr.h

Purpose

To start Timer A.

Description

If no period is set, the counter starts with the highest counter value (0xFFFF FFFF).

Tip

Use `srtk_timerA_period_set` to set the period.

Return value

None

Related topics

References

[srtk_timerA_period_set](#).....67

srtk_timerA_stop

Syntax

```
void srtk_timerA_stop(void)
```

Include file

SrtkTmr.h

Purpose

To stop Timer A.

Tip

Use `srtk_timerA_start` to resume from the current value.

Return value None

Related topics References

[srtk_timerA_start.....](#) 69

Timer B

Introduction

Timer B is a counter generating an interrupt when it reaches its compare value and continues counting. Thus, Timer B is designed only for single timer events. If your model requires periodic timer events, use Timer A (refer to [Timer A](#) on page 65). If Timer A is already used, use Timer B and set its compare value periodically (function `srtk_timerB_compare_set_periodically`).

For further information on Timer B, refer to [Timer Services \(MicroLabBox Features\)](#).

Where to go from here

Information in this section

Example of Using Timer B Functions	72
Gives you an example how to use Timer B.	
srtk_timerB_init	72
To initialize Timer B.	
srtk_timerB_compare_set	73
To set the new compare value.	
srtk_timerB_compare_set_periodically	74
To periodically set a new compare value.	
srtk_timerB_read	75
To read the current value of Timer B.	
srtk_timerB_start	75
To start Timer B.	
srtk_timerB_stop	76
To stop Timer B.	

Information in other sections

For information on handling Timer B interrupts	
Timer Interrupt Control	82
Interrupt Handling	93

Example of Using Timer B Functions

Example

The following example demonstrates how to use Timer B functions.

```
#include <Brtenv.h>
#define DT 1e-4           /* 100 µs simulation step size */
/* ++ variables for execution time profiling ++++++ */
Float64 exec_time;           /* execution time */
Float64 timerB;              /* timerB read value */
/* ++ adjust values for timerB ++++++ */
Float64 upc_period = .001;    /* upcounter period in sec */
UInt16 scale_value = 2;       /* set the scaling of timerB */
/* ++ counter for Interrupt service functions ++++++ */
Int32 timerB_counter = 0;
void isr_timerB(void)
{
    ts_timestamp_type ts;
    srtk_begin_isr_timerB();           /* overrun check */
    RTLlib_TIC_START();
    timerB_counter++;                /* counter for timerB interrupts */
    srtk_timerB_read(&timerB);        /* read timerB */
    ts_timestamp_read(&ts);
    DsDaq_Service(0, 0, 1,
                  (DsDaqSTimestampStruct *)&ts); /* data acquisition service */
    exec_time = RTLlib_TIC_READ();
    srtk_end_isr_timerB();           /* overrun check */
}
void main(void)
{
    /* init processor board */
    init();
    /* periodic event with TimerB */
    srtk_start_isr_timerB(scale_value,
                          upc_period,
                          isr_timerB);
    /* Background task */
    while(1)
    {
        RTLlib_BACKGROUND_SERVICE(); /* host PC service */
    }
}
```

srtk_timerB_init

Syntax

```
void srtk_timerB_init(UInt16 scale)
```

Include file

`SrtkTmr.h`

Purpose To initialize Timer B.

Parameters

scale	Specifies a value within the range 0 ... 7 that defines the prescaler setting of Timer B as a function of the I/O bus clock. The I/O bus clock runs at a speed of 100 MHz (10 ns).
--------------	--

Scale Value	Timer B Clock/ Bus Clock	(I/O bus clock is 100 MHz)	
		Timer B Clock	Prescaler Period
0	1/4	25.0 MHz	40 ns
1	1/8	12.5 MHz	80 ns
2	1/16	6.25 MHz	160 ns
3	1/32	3.125 MHz	320 ns
4	1/64	1.5625 MHz	640 ns
5	1/128	0.78125 MHz	1280 ns
6	1/256	0.390625 MHz	2560 ns
7	1/512	0.1953125 MHz	5120 ns

Return value None

Example The MicroLabBox I/O bus clock has a resolution of 10 ns. To achieve a timer period of 160 ns, the prescaler must be set to 1/16:

```
srtk_timerB_init(SRTK_TIMERB_1_16_BCLK);
```

Related topics

References

srtk_timerB_compare_set.....	73
------------------------------	----

srtk_timerB_compare_set

Syntax

void srtk_timerB_compare_set(Float64 delta_time)
--

Include file SrtkTmr.h

Purpose To set the new compare value.

Description	The compare value to be written to the Timer B compare register is calculated by adding the <code>delta_time</code> to the current timer value. When the counter value matches the value of the compare register, Timer B generates an interrupt. To make the Timer B interrupt available, refer to Timer Interrupt Control on page 82 and Interrupt Handling on page 93. If you want to generate a Timer B interrupt periodically, use the function <code>srtk_timerB_compare_set_periodically</code> .
Parameters	<code>delta_time</code> Specifies the period in seconds.
Return value	None
Related topics	References <code>srtk_timerB_compare_set_periodically</code> 74 <code>srtk_timerB_init</code> 72

[srtk_timerB_compare_set_periodically](#)

Syntax	<pre>void srtk_timerB_compare_set_periodically(Float64 delta_time)</pre>
Include file	<code>SrtkTmr.h</code>
Purpose	To periodically set a new compare value.
Description	This function is used in the Timer B interrupt service routine to make Timer B a periodic timer. The new compare value to be written to the Timer B compare register is calculated by adding the <code>delta_time</code> to the old compare value. When the counter value matches the value of the compare register, Timer B generates an interrupt. This function is automatically called in your interrupt service routine when using <code>srtk_begin_isr_timerB</code> . To make the Timer B interrupt available, refer to Timer Interrupt Control on page 82 and Interrupt Handling on page 93.

Parameters **delta_time** Specifies the period in seconds.

Return value None

Related topics **References**

srtk_begin_isr_timerA.....	83
srtk_timerB_compare_set.....	73

srtk_timerB_read

Syntax `void srtk_timerB_read(Float64 *time)`

Include file SrtkTmr.h

Purpose To read the current value of Timer B.

Parameters **time** Specifies the pointer to the current value of Timer B. The value is given in seconds.

Return value None

Related topics **Examples**

Example of Using Timer B Functions.....	72
---	----

srtk_timerB_start

Syntax `void srtk_timerB_start(void)`

Include file SrtkTmr.h

Purpose	To start Timer B.
---------	-------------------

Tip

Use `srtk_timerB_compare_set` to set the compare value.

Return value	None
--------------	------

Related topics	References
----------------	------------

<code>srtk_timerB_compare_set</code>	73
<code>srtk_timerB_stop</code>	76

[srtk_timerB_stop](#)

Syntax	<code>void srtk_timerB_stop(void)</code>
--------	--

Include file	<code>SrtkTmr.h</code>
--------------	------------------------

Purpose	To stop Timer B.
---------	------------------

Tip

Use `srtk_timerB_start` to continue.

Return value	None
--------------	------

Related topics	References
----------------	------------

<code>srtk_timerB_start</code>	75
--------------------------------------	----

Timer D

Introduction

Timer D is functionally identical to Timer A. Timer D is a down counter generating an interrupt whenever it reaches zero. The period value is then reloaded automatically.

For further information on Timer D, refer to [Timer Services \(MicroLabBox Features\)](#).

Where to go from here

Information in this section

Example of Using Timer D functions	77
Gives you an example how to use Timer D.	
srtk_timerD_period_set	78
To define the period of Timer D.	
srtk_timerD_period_reload	79
To set a new period of Timer D and restart it immediately.	
srtk_timerD_read	79
To read the current value of Timer D.	
srtk_timerD_start	80
To start Timer D.	
srtk_timerD_stop	81
To stop Timer D.	

Information in other sections

For information on handling Timer D interrupts.	
Timer Interrupt Control	82
Interrupt Handling	93

Example of Using Timer D functions

Example

The following example demonstrates how to use Timer D functions.

```
#include <Brtenv.h>
#define DT 1.0e-4           /* 100 µs simulation step size */
/*-- variables for host PC -----
Float64 exec_time, timeD;      /* execution time */
```

```

void ad_routine(void)
{
    ts_timestamp_type ts;
    Float64 old_timeD;
    srtk_begin_isr_timerD(); /* overrun check TimerD */
    srtk_timerD_read(&old_timeD);
    ts_timestamp_read(&ts);
    DsDaq_Service(0, 0, 1,
                  (DsDaqSTimestampStruct *) &ts); /* data acquisition service */
    /*--- do something ---*/
    srtk_timerD_read(&timeD);
    exec_time = old_timeD - timeD; /* exec time with Timer D */
    srtk_end_isr_timerD(); /* overrun check TimerD */
}
void main(void)
{
    /* init processor board */
    init();
    /* set period of timerD */
    timeD = DT;
    srtk_timerD_period_set(timeD);
    /* periodic event in ISR */
    srtk_start_isr_timerD(timeD, ad_routine);
    /* Background tasks */
    while(1)
    {
        RTLIB_BACKGROUND_SERVICE(); /* host PC service */
    }
}

```

srtk_timerD_period_set

Syntax

void srtk_timerD_period_set(Float64 time)

Include file

SrtkTmr.h

Purpose

To define the period of Timer D.

Description

If the timer is not running, the new value is loaded immediately. If the timer is running, the new value is loaded the next time the timer reaches zero.

Parameters

time Specifies the period in seconds.

Return value

None

Related topics**Examples**

[Example of Using Timer D functions](#).....77

References

[srtk_timerD_period_reload](#).....79

srtk_timerD_period_reload

Syntax

```
void srtk_timerD_period_reload(Float64 time)
```

Include file

SrtkTmr.h

Purpose

To set a new period of Timer D and restart it immediately.

Description

The new value is loaded immediately. Timer D is stopped, the new value is set, and Timer D is started again.

Parameters

time Specifies the period in seconds.

Return value

None

Related topics**References**

[srtk_timerD_period_set](#).....78

srtk_timerD_read

Syntax

```
void srtk_timerD_read(Float64 *time)
```

Include file

SrtkTmr.h

Purpose	To read the current value of Timer D.
Parameters	time Specifies the pointer to the current value of Timer D. The value is stated in seconds.
Return value	None
Related topics	Examples Example of Using Timer D functions..... 77

srtk_timerD_start

Syntax	<code>void srtk_timerD_start(void)</code>
Include file	<code>SrtkTmr.h</code>
Purpose	To start Timer D.
Description	If no period is set, the counter starts with the highest counter value (0xFFFF FFFF).
	Tip Use <code>srtk_timerD_period_set</code> to set the period.
Return value	None
Related topics	References srtk_timerD_period_set..... 78

srtk_timerD_stop

Syntax

```
void srtk_timerD_stop(void)
```

Include file

`SrtkTmr.h`

Purpose

To stop Timer D.

Tip

Use `srtk_timerD_start` to resume from the current value.

Return value

None

Related topics**References**

[srtk_timerD_start](#).....80

Timer Interrupt Control

Introduction

These functions are used to install interrupt service routines for the available timers and to perform overrun checks for the defined interrupt service routines.

Tip

Here you find the descriptions of platform-specific functions and generic `RTLIB_SRT_XXX` macros. It is recommended to use the generic macros if available.

Where to go from here

Information in this section

srtk_begin_isr_timerA	83
To check for an overrun in the interrupt service routine assigned to Timer A.	
srtk_begin_isr_timerB	84
To check for an overrun in the interrupt service routine assigned to Timer B.	
srtk_begin_isr_timerD	85
To check for an overrun in the interrupt service routine assigned to Timer D.	
srtk_end_isr_timerA	85
To check for an overrun in the interrupt service routine assigned to Timer A.	
srtk_end_isr_timerB	86
To check for an overrun in the interrupt service routine assigned to Timer B.	
srtk_end_isr_timerD	86
To check for an overrun in the interrupt service routine assigned to Timer D.	
srtk_start_isr_timerA	87
To install an interrupt service routine for Timer A.	
srtk_start_isr_timerB	88
To install an interrupt service routine for Timer B.	
srtk_start_isr_timerD	89
To install an interrupt service routine for Timer D.	
RTLIB_SRT_ISR_BEGIN	90
To check for an overrun in the interrupt service routine assigned to Timer A.	

[RTLIB_SRT_ISR_END](#).....91
To check the overrun in the interrupt service routine assigned to Timer A.

[RTLIB_SRT_START](#).....91
To install an interrupt service routine for Timer A.

Information in other sections

[Interrupt Handling](#).....93

srtk_begin_isr_timerA

Syntax `srtk_begin_isr_timerA()`

Include file `SrtkTmrInt.h`

Purpose To check for an overrun in the interrupt service routine assigned by `srtk_start_isr_timerA`.

Description When the execution time of the interrupt service routine exceeds the interrupt period (overrun), the interrupt is stopped, and an appropriate error message is generated.

Return value None

Example This example shows an interrupt service routine with overrun check:

```
void timerA_interrupt(void)
{
    srtk_begin_isr_timerA();
    /* interrupt service routine */
    srtk_end_isr_timerA();
}
```

Related topics**References**

RTLIB_SRT_ISR_BEGIN.....	90
srtk_end_isr_timerA.....	85

srtk_begin_isr_timerB

Syntax

```
srtk_begin_isr_timerB()
```

Include file

SrtkTmrInt.h

Purpose

To check for an overrun in the interrupt service routine assigned by **srtk_start_isr_timerB** and to reload the compare value.

Description

When the execution time of the interrupt service routine exceeds the interrupt period (overrun), the interrupt is stopped, and an appropriate error message is generated.

Return value

None

Example

This example shows an interrupt service routine with overrun check:

```
void timerB_interrupt(void)
{
    srtk_begin_isr_timerB();
    /* interrupt service routine */
    srtk_end_isr_timerB();
}
```

Related topics**References**

srtk_end_isr_timerB.....	86
srtk_start_isr_timerB.....	88

srtk_begin_isr_timerD

Syntax	<code>srtk_begin_isr_timerD()</code>
Include file	<code>SrtkTmrInt.h</code>
Purpose	To check for an overrun in the interrupt service routine assigned by <code>srtk_start_isr_timerD</code> .
Description	When the execution time of the interrupt service routine exceeds the interrupt period (overrun), the interrupt is stopped, and an appropriate error message is generated.
Return value	None
Example	This example shows an interrupt service routine with overrun check:
	<pre>void timerD_interrupt(void) { srtk_begin_isr_timerD(); /* interrupt service routine */ srtk_end_isr_timerD(); }</pre>
Related topics	References
	<p>srtk_end_isr_timerD.....86</p>

srtk_end_isr_timerA

Syntax	<code>srtk_end_isr_timerA()</code>
Include file	<code>SrtkTmrInt.h</code>
Purpose	To check for an overrun in the interrupt service routine assigned by <code>srtk_start_isr_timerA</code> .

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

RTLIB_SRT_ISR_END.....	91
srtk_begin_isr_timerA.....	83
srtk_start_isr_timerA.....	87

srtk_end_isr_timerB

Syntax	<code>srtk_end_isr_timerB()</code>
---------------	------------------------------------

Include file	<code>SrtkTmrInt.h</code>
---------------------	---------------------------

Purpose	To check for an overrun in the interrupt service routine assigned by <code>srtk_start_isr_timerB</code> .
----------------	--

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

srtk_begin_isr_timerB.....	84
srtk_start_isr_timerB.....	88

srtk_end_isr_timerD

Syntax	<code>srtk_end_isr_timerD()</code>
---------------	------------------------------------

Include file	<code>SrtkTmrInt.h</code>
---------------------	---------------------------

Purpose	To check for an overrun in the interrupt service routine assigned by <code>srtk_start_isr_timerD</code> .
----------------	--

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

srtk_begin_isr_timerD.....	85
srtk_start_isr_timerD.....	89

srtk_start_isr_timerA

Syntax

```
void srtk_start_isr_timerA(
    Float64 sampling_period,
    Srtk_Int_Handler_Type isr_function_name)
```

Include file

SrtkTmrInt.h

Purpose

To install `isr_function_name` as an interrupt service routine for Timer A.

Description

The function sets the period of Timer A, installs the specified routine as interrupt handler, and starts Timer A.

If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use `srtk_begin_isr_timerA` and `srtk_end_isr_timerA` in your interrupt service routine to install an overrun check.

Parameters

`sampling_period` Specifies the period in seconds.

`isr_function_name` Specifies the name of the function to be assigned to the Timer A interrupt. This function must not have an input parameter or a return value, i.e., `void isr_function_name(void)`.

Return value

None

Example

This example installs the function `timerA_interrupt`, which is called when the Timer A interrupt occurs, namely every 20 µs:

```
srtk_start_isr_timerA(20e-6, timerA_interrupt);
```

Related topics

References

RTLIB_SRT_START.....	91
srtk_begin_isr_timerA.....	83
srtk_end_isr_timerA.....	85

[srtk_start_isr_timerB](#)

Syntax

```
void srtk_start_isr_timerB(
    UInt32 scale,
    Float64 sampling_period,
    Srtk_Int_Handler_Type isr_function_name)
```

Include file

SrtkTmrInt.h

Purpose

To install `isr_function_name` as an interrupt service routine for Timer B and initialize Timer B.

Description

The function sets the compare value of Timer B, installs the specified routine as interrupt handler, and starts Timer B. Because Timer B is not a periodic timer, you must use `srtk_begin_isr_timerB` and `srtk_end_isr_timerB` in your interrupt service routine to reload the compare value. In addition, you install an overrun check that prevents the execution time of the interrupt service routine from exceeding the interrupt period.

Parameters

scale Specifies a value within the range 0 ... 7 that defines the prescaler setting of Timer B as a function of the I/O bus clock. The I/O bus clock runs at a speed of 100 MHz (10 ns).

Scale Value	Timer B Clock/ Bus Clock	(I/O bus clock is 100 MHz)	
		Timer B Clock	Prescaler Period
0	1/4	25.0 MHz	40 ns
1	1/8	12.5 MHz	80 ns
2	1/16	6.25 MHz	160 ns
3	1/32	3.125 MHz	320 ns
4	1/64	1.5625 MHz	640 ns
5	1/128	0.78125 MHz	1280 ns

Scale Value	Timer B Clock/ Bus Clock	(I/O bus clock is 100 MHz)	
		Timer B Clock	Prescaler Period
6	1/256	0.390625 MHz	2560 ns
7	1/512	0.1953125 MHz	5120 ns

sampling_period Specifies the period in seconds.

isr_function_name Specifies the name of the function to be assigned to the Timer B interrupt. This function must not have an input parameter or a return value, i.e., `void isr_function_name(void)`.

Return value None

Example This example installs the function `timerB_interrupt`, which is called when the Timer B interrupt occurs, namely every 100 µs:

```
srtk_start_isr_timerB(0, 100e-6, timerB_interrupt)
```

Related topics References

<code>srtk_begin_isr_timerB...</code>	84
<code>srtk_end_isr_timerB...</code>	86

srtk_start_isr_timerD

Syntax

```
void srtk_start_isr_timerD(
    Float64 sampling_period,
    Srtk_Int_Handler_Type isr_function_name)
```

Include file SrtkTmrInt.h

Purpose To install `isr_function_name` as an interrupt service routine for Timer D.

Description The function sets the period of Timer D, installs the specified routine as interrupt handler, and starts Timer D.

If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use `srtk_begin_isr_timerD` and `srtk_end_isr_timerD` in your interrupt service routine to install an overrun check.

Parameters	sampling_period Specifies the period in seconds. isr_function_name Specifies the name of the function to be assigned to the Timer D interrupt. This function must not have an input parameter or a return value, i.e., <code>void isr_function_name(void)</code> .				
Return value	None				
Example	This example installs the function <code>timerD_interrupt</code> , which is called when the Timer D interrupt occurs, namely every 20 µs: <pre>srtk_start_isr_timerD(20e-6, timerD_interrupt);</pre>				
Related topics	References <table><tr><td>srtk_begin_isr_timerD.....</td><td>85</td></tr><tr><td>srtk_end_isr_timerD.....</td><td>86</td></tr></table>	srtk_begin_isr_timerD.....	85	srtk_end_isr_timerD.....	86
srtk_begin_isr_timerD.....	85				
srtk_end_isr_timerD.....	86				

RTLIB_SRT_ISR_BEGIN

Syntax	<code>RTLIB_SRT_ISR_BEGIN()</code>
Include file	<code>SrtkStd.h</code>
Purpose	To check for an overrun in the interrupt service routine assigned by <code>RTLIB_SRT_START</code> .
Description	When the execution time of the interrupt service routine exceeds the interrupt period (overrun), the interrupt is stopped, and an appropriate error message is generated.
Return value	None

Example

This example shows an interrupt service routine with overrun check:

```
void timerA_interrupt(void)
{
    RTLIB_SRT_ISR_BEGIN();
    /* interrupt service routine */
    RTLIB_SRT_ISR_END();
}
```

Related topics**References**

RTLIB_SRT_START.....	91
srtk_begin_isr_timerA.....	83

RTLIB_SRT_ISR_END

Syntax

```
RTLIB_SRT_ISR_END()
```

Include file

SrtkStd.h

Purpose

To check for an overrun in the interrupt service routine assigned by **RTLIB_SRT_START**.

Return value

None

Related topics**References**

RTLIB_SRT_START.....	91
srtk_end_isr_timerA.....	85

RTLIB_SRT_START

Syntax

```
RTLIB_SRT_START(
    Float64 sampling_period,
    Srtk_Int_Handler_Type isr_function_name)
```

Include file	SrtkStd.h
Purpose	To install <code>isr_function_name</code> as an interrupt service routine for Timer A.
Description	<p>The function sets the period of Timer A, installs the specified routine as interrupt handler, and starts Timer A.</p> <p>If the execution time of the interrupt service routine exceeds the interrupt period, an overrun occurs. Use <code>RTLIB_SRT_ISR_BEGIN</code> and <code>RTLIB_SRT_ISR_END</code> in your interrupt service routine to install an overrun check.</p>
Parameters	<p><code>sampling_period</code> Specifies the period in seconds.</p> <p><code>isr_function_name</code> Specifies the name of the function to be assigned to the Timer A interrupt. This function must not have an input parameter or a return value, i.e., <code>void isr_function_name(void)</code>.</p>
Return value	None
Example	This example installs the function <code>timerA_interrupt</code> , which is called when the Timer A interrupt occurs, namely every 20 µs: <pre>RTLIB_SRT_START(20e-6, timerA_interrupt);</pre>
Related topics	References
	srtk_start_isr_timerA87

Interrupt Handling

Introduction

Use the interrupt handling functions to make interrupts available as trigger sources. If you want to use an interrupt, you have to install an appropriate handler and enable interrupt handling. The interrupt handling uses the interrupt identification (IntId) to identify the interrupt handler that has been installed for this interrupt. Whether or not an interrupt has been generated is indicated by the interrupt flag.

Note

For examples of the installation of interrupt service routines for the Timer A, Timer B and Timer D interrupts, refer to [srtk_set_interrupt_vector](#) on page 104 and [Timer Interrupt Control](#) on page 82.

For further information on the interrupt handling, refer to [Interrupt Handling \(MicroLabBox Features\)](#).

Interrupt service routine type

The interrupt service routine type is defined as follows:

```
typedef void (*Srtk_Int_Handler_Type)(void)
```

Where to go from here

Information in this section

srtk_disable_hardware_int	94
To disable the specified hardware interrupt when the interrupts are still globally enabled.	
srtk_disable_hardware_int_bm	95
To disable several hardware interrupts when the interrupts are still globally enabled.	
srtk_enable_hardware_int	96
To enable the specified hardware interrupt.	
srtk_enable_hardware_int_bm	97
To enable several hardware interrupts.	
srtk_get_interrupt_flag	99
To get the interrupt flag for the specified interrupt.	
srtk_get_interrupt_flag_bm	100
To get the interrupt flag for several interrupts.	
srtk_get_interrupt_vector	101
To get the address of the interrupt service routine related to the given interrupt.	
srtk_reset_interrupt_flag	102
To reset the interrupt flag for the specified interrupt.	

srtk_reset_interrupt_flag_bm	103
To reset the interrupt flag for several interrupts.	
srtk_set_interrupt_vector	104
To install an interrupt service routine for the selected interrupt.	
RTLIB_INT_DISABLE	106
To globally disable the interrupts.	
RTLIB_INT_ENABLE	106
To globally enable the interrupts.	
RTLIB_INT_RESTORE	107
To restore the previous state.	
RTLIB_INT_SAVE_AND_DISABLE	107
To disable the interrupts globally and save the state.	
RTLIB_SRT_DISABLE	108
To disable the hardware interrupt for the sampling rate timer when the interrupts are still globally enabled.	
RTLIB_SRT_ENABLE	109
To enable the hardware interrupt for the sampling rate timer.	

srtk_disable_hardware_int

Syntax

```
void srtk_disable_hardware_int(UInt32 IntID)
```

Include file

SrtkInt.h

Purpose

To disable the specified hardware interrupt when the interrupts are still globally enabled (see [RTLIB_INT_ENABLE](#)).

Description

This function sets the corresponding bit of the Interrupt Mask Register (IMR).

Parameters

IntID Specifies the interrupt that is to be disabled.

The following symbols are predefined:

Predefined Symbol	Meaning
SRTK_INT_TIMER_A	Timer A interrupt
SRTK_INT_TIMER_B	Timer B interrupt

Predefined Symbol	Meaning
SRTK_INT_TIMER_D	Timer D interrupt
SRTK_INT_GL_0	Gigalink 0 interrupt
...	...
SRTK_INT_GL_3	Gigalink 3 interrupt
SRTK_INT_MACROTICK	Macrotick interrupt
SRTK_INT_FWD_TIMER_A	Forwarded Timer A interrupt
SRTK_INT_FWD_TIMER_B	Forwarded Timer B interrupt
SRTK_INT_IO_ETH	I/O Ethernet interrupt

Note

Level-triggered interrupts have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value None

Related topics References

RTLIB_INT_DISABLE.....	106
srtk_disable_hardware_int_bm.....	95
srtk_enable_hardware_int.....	96

srtk_disable_hardware_int_bm

Syntax `void srtk_disable_hardware_int_bm(UINT32 IntMask)`

Include file SrtkInt.h

Purpose To disable several hardware interrupts when the interrupts are still globally enabled (see **RTLIB_INT_ENABLE**).

Description This function sets the corresponding bit of the Interrupt Mask Register (IMR).

Parameters

IntMask Specifies the interrupts that are to be disabled. To specify more than one interrupt, you can combine the predefined symbols using the logical operator OR.

The following symbols are predefined:

Predefined Symbol	Meaning
SRTK_INT_MASK_TIMER_A	Timer A interrupt
SRTK_INT_MASK_TIMER_B	Timer B interrupt
SRTK_INT_MASK_TIMER_D	Timer D interrupt
SRTK_INT_MASK_GL_0	Gigalink 0 interrupt
...	...
SRTK_INT_MASK_GL_3	Gigalink 3 interrupt
SRTK_INT_MASK_MACROTICK	Macrotick interrupt
SRTK_INT_MASK_FWD_TIMER_A	Forwarded Timer A interrupt
SRTK_INT_MASK_FWD_TIMER_B	Forwarded Timer B interrupt
SRTK_INT_MASK_IO_ETH	I/O Ethernet interrupt

Note

Level-triggered interrupts have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value None

Related topics References

RTLIB_INT_ENABLE.....	106
srtk_enable_hardware_int_bm.....	97

srtk_enable_hardware_int

Syntax `void srtk_enable_hardware_int(UINT32 IntID)`

Include file SrtkInt.h

Purpose To enable the specified hardware interrupt.

Description	This function only clears the corresponding bit of the Interrupt Mask Register (IMR). However, the specified hardware interrupt is available only when the interrupts are globally enabled (see RTLIB_INT_ENABLE).
--------------------	---

Parameters	IntID Specifies the interrupt that is to be enabled.
-------------------	---

The following symbols are predefined:

Predefined Symbol	Meaning
SRTK_INT_TIMER_A	Timer A interrupt
SRTK_INT_TIMER_B	Timer B interrupt
SRTK_INT_TIMER_D	Timer D interrupt
SRTK_INT_GL_0	Gigalink 0 interrupt
...	...
SRTK_INT_GL_3	Gigalink 3 interrupt
SRTK_INT_MACROTICK	Macrotick interrupt
SRTK_INT_FWD_TIMER_A	Forwarded Timer A interrupt
SRTK_INT_FWD_TIMER_B	Forwarded Timer B interrupt
SRTK_INT_IO_ETH	I/O Ethernet interrupt

Note

Level-triggered interrupts have to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

RTLIB_INT_ENABLE.....	106
srtk_disable_hardware_int.....	94
srtk_enable_hardware_int_bm.....	97

srtk_enable_hardware_int_bm

Syntax	<code>void srtk_enable_hardware_int_bm(UINT32 IntMask)</code>
---------------	---

Include file	SrtkInt.h																						
Purpose	To enable several hardware interrupts.																						
Description	This function clears the corresponding bits of the Interrupt Mask Register (IMR). However, the specified hardware interrupts are available only when the interrupts are globally enabled (see RTLIB_INT_ENABLE).																						
Parameters	<p>IntMask Specifies the interrupts that are to be enabled. To specify more than one interrupt, you can combine the predefined symbols by using the logical operator OR.</p> <p>The following symbols are predefined:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>SRTK_INT_MASK_TIMER_A</td><td>Timer A interrupt</td></tr> <tr> <td>SRTK_INT_MASK_TIMER_B</td><td>Timer B interrupt</td></tr> <tr> <td>SRTK_INT_MASK_TIMER_D</td><td>Timer D interrupt</td></tr> <tr> <td>SRTK_INT_MASK_GL_0</td><td>Gigalink 0 interrupt</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td>SRTK_INT_MASK_GL_3</td><td>Gigalink 3 interrupt</td></tr> <tr> <td>SRTK_INT_MASK_MACROTICK</td><td>Macrotick interrupt</td></tr> <tr> <td>SRTK_INT_MASK_FWD_TIMER_A</td><td>Forwarded Timer A interrupt</td></tr> <tr> <td>SRTK_INT_MASK_FWD_TIMER_B</td><td>Forwarded Timer B interrupt</td></tr> <tr> <td>SRTK_INT_MASK_IO_ETH</td><td>I/O Ethernet interrupt</td></tr> </tbody> </table>	Predefined Symbol	Meaning	SRTK_INT_MASK_TIMER_A	Timer A interrupt	SRTK_INT_MASK_TIMER_B	Timer B interrupt	SRTK_INT_MASK_TIMER_D	Timer D interrupt	SRTK_INT_MASK_GL_0	Gigalink 0 interrupt	SRTK_INT_MASK_GL_3	Gigalink 3 interrupt	SRTK_INT_MASK_MACROTICK	Macrotick interrupt	SRTK_INT_MASK_FWD_TIMER_A	Forwarded Timer A interrupt	SRTK_INT_MASK_FWD_TIMER_B	Forwarded Timer B interrupt	SRTK_INT_MASK_IO_ETH	I/O Ethernet interrupt
Predefined Symbol	Meaning																						
SRTK_INT_MASK_TIMER_A	Timer A interrupt																						
SRTK_INT_MASK_TIMER_B	Timer B interrupt																						
SRTK_INT_MASK_TIMER_D	Timer D interrupt																						
SRTK_INT_MASK_GL_0	Gigalink 0 interrupt																						
...	...																						
SRTK_INT_MASK_GL_3	Gigalink 3 interrupt																						
SRTK_INT_MASK_MACROTICK	Macrotick interrupt																						
SRTK_INT_MASK_FWD_TIMER_A	Forwarded Timer A interrupt																						
SRTK_INT_MASK_FWD_TIMER_B	Forwarded Timer B interrupt																						
SRTK_INT_MASK_IO_ETH	I/O Ethernet interrupt																						
	<p>Note</p> <p>Level-triggered interrupts have to be acknowledged in the interrupt service routine before they are enabled globally again.</p>																						
Return value	None																						
Related topics	References <table border="0"> <tr> <td style="padding-right: 20px;">RTLIB_INT_ENABLE.....</td> <td>106</td> </tr> <tr> <td>srtk_disable_hardware_int_bm.....</td> <td>95</td> </tr> </table>	RTLIB_INT_ENABLE	106	srtk_disable_hardware_int_bm	95																		
RTLIB_INT_ENABLE	106																						
srtk_disable_hardware_int_bm	95																						

srtk_get_interrupt_flag

Syntax

```
int srtk_get_interrupt_flag(UINT32 IntID)
```

Include file

`SrtkInt.h`

Purpose

To get the interrupt flag for the specified interrupt.

Description

The interrupt flag indicates whether or not the specified interrupt has been generated.

Parameters

IntID Specifies the interrupt whose interrupt flag is to be read.

The following symbols are predefined:

Predefined Symbol	Meaning
<code>SRTK_INT_TIMER_A</code>	Timer A interrupt
<code>SRTK_INT_TIMER_B</code>	Timer B interrupt
<code>SRTK_INT_TIMER_D</code>	Timer D interrupt
<code>SRTK_INT_GL_0</code>	Gigalink 0 interrupt
...	...
<code>SRTK_INT_GL_3</code>	Gigalink 3 interrupt
<code>SRTK_INT_MACROTICK</code>	Macrotick interrupt
<code>SRTK_INT_FWD_TIMER_A</code>	Forwarded Timer A interrupt
<code>SRTK_INT_FWD_TIMER_B</code>	Forwarded Timer B interrupt
<code>SRTK_INT_IO_ETH</code>	I/O Ethernet interrupt

Note

Level-triggered interruptshave to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value

This function returns the value of the interrupt flag:

Value	Meaning
0	Interrupt has not been generated
1	Interrupt has been generated

Related topics

References

srtk_get_interrupt_flag_bm	100
--	-------	-----

[srtk_get_interrupt_flag_bm](#)

Syntax

<code>int srtk_get_interrupt_flag_bm(UINT32 flag)</code>
--

Include file

<code>SrtkInt.h</code>

Purpose

To get the interrupt flag for several interrupts.

Description

The interrupt flag indicates whether or not one of the specified interrupts has been generated.

Parameters

flag Specifies a bitmask of interrupts that are to be checked. To specify more than one interrupt, you can combine the predefined symbols using the logical operator OR.

The following symbols are predefined:

Predefined Symbol	Meaning
<code>SRTK_INT_TIMER_A</code>	Timer A interrupt
<code>SRTK_INT_TIMER_B</code>	Timer B interrupt
<code>SRTK_INT_TIMER_D</code>	Timer D interrupt
<code>SRTK_INT_GL_0</code>	Gigalink 0 interrupt
...	...
<code>SRTK_INT_GL_3</code>	Gigalink 3 interrupt
<code>SRTK_INT_MACROTICK</code>	Macrotick interrupt
<code>SRTK_INT_FWD_TIMER_A</code>	Forwarded Timer A interrupt
<code>SRTK_INT_FWD_TIMER_B</code>	Forwarded Timer B interrupt
<code>SRTK_INT_IO_ETH</code>	I/O Ethernet interrupt

Note

Level-triggered interruptshave to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value

This function returns the value of the interrupt flag:

Value	Meaning
0	Interrupt has not been generated
1	At least one interrupt has been generated

Related topics**References**

[srtk_get_interrupt_flag](#).....99

srtk_get_interrupt_vector

Syntax

```
Srtk_Int_Handler_Type srtk_get_interrupt_vector(
    UInt32 IntID)
```

Include file

SrtkInt.h

Purpose

To get the address of the interrupt service routine related to the given interrupt.

Description

Use this function to retrieve the interrupt service routine installed for a given interrupt source. If no user handler has been installed, the default handler will be returned.

Parameters

IntID Specifies the interrupt source for which the installed handler is to be returned.

The following symbols are predefined:

Predefined Symbol	Meaning
SRTK_INT_TIMER_A	Timer A interrupt
SRTK_INT_TIMER_B	Timer B interrupt

Predefined Symbol	Meaning
SRTK_INT_TIMER_D	Timer D interrupt
SRTK_INT_GL_0	Gigalink 0 interrupt
...	...
SRTK_INT_GL_3	Gigalink 3 interrupt
SRTK_INT_MACROTICK	Macrotick interrupt
SRTK_INT_FWD_TIMER_A	Forwarded Timer A interrupt
SRTK_INT_FWD_TIMER_B	Forwarded Timer B interrupt
SRTK_INT_IO_ETH	I/O Ethernet interrupt

Return value

This function returns the address of the interrupt service routine that is installed for this interrupt.

srtk_reset_interrupt_flag

Syntax

```
void srtk_reset_interrupt_flag(UINT32 IntID)
```

Include file

SrtkInt.h

Purpose

To reset the interrupt flag for the specified interrupt.

Parameters

IntID Specifies the interrupt for which the interrupt flag is to be reset.
The following symbols are predefined:

Predefined Symbol	Meaning
SRTK_INT_TIMER_A	Timer A interrupt
SRTK_INT_TIMER_B	Timer B interrupt
SRTK_INT_TIMER_D	Timer D interrupt
SRTK_INT_GL_0	Gigalink 0 interrupt
...	...
SRTK_INT_GL_3	Gigalink 3 interrupt
SRTK_INT_MACROTICK	Macrotick interrupt
SRTK_INT_FWD_TIMER_A	Forwarded Timer A interrupt
SRTK_INT_FWD_TIMER_B	Forwarded Timer B interrupt
SRTK_INT_IO_ETH	I/O Ethernet interrupt

Note

Level-triggered interruptshave to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

srtk_reset_interrupt_flag_bm	103
--	-----

srtk_reset_interrupt_flag_bm

Syntax	<code>void srtk_reset_interrupt_flag_bm(UINT32 flag)</code>
---------------	---

Include file	<code>SrtkInt.h</code>
---------------------	------------------------

Purpose	To reset the interrupt flag for several interrupts.
----------------	---

Parameters	flag Specifies the bitmask of interrupts whose interrupt flag is to be reset. To specify more than one interrupt, you can combine the predefined symbols using the logical operator OR.
-------------------	--

The following symbols are predefined:

Predefined Symbol	Meaning
<code>SRTK_INT_MASK_TIMER_A</code>	Timer A interrupt
<code>SRTK_INT_MASK_TIMER_B</code>	Timer B interrupt
<code>SRTK_INT_MASK_TIMER_D</code>	Timer D interrupt
<code>SRTK_INT_MASK_GL_0</code>	Gigalink 0 interrupt
...	...
<code>SRTK_INT_MASK_GL_3</code>	Gigalink 3 interrupt
<code>SRTK_INT_MASK_MACROTICK</code>	Macrotick interrupt
<code>SRTK_INT_MASK_FWD_TIMER_A</code>	Forwarded Timer A interrupt
<code>SRTK_INT_MASK_FWD_TIMER_B</code>	Forwarded Timer B interrupt
<code>SRTK_INT_MASK_IO_ETH</code>	I/O Ethernet interrupt

Note

Level-triggered interruptshave to be acknowledged in the interrupt service routine before they are enabled globally again.

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

srtk_reset_interrupt_flag	102
---	-----

srtk_set_interrupt_vector

Syntax

```
Srtk_Int_Handler_Type srtk_set_interrupt_vector(
    UInt32 IntID,
    Srtk_Int_Handler_Type Handler)
```

Include file

SrtkInt.h

Purpose

To install an interrupt service routine for the selected interrupt.

Note

- When you want to migrate your code written for DS1103 to MicroLabBox, you have to note that the `SaveRegs` parameter is no more available.
- Use `RTLIB_INT_ENABLE` to enable interrupts.
- The installation of interrupt service routines for the Timer A, Timer B, and Timer D interrupts is different from that of other interrupts. Refer to the example below and to [Timer Interrupt Control](#) on page 82.

Parameters

IntID Identifies the interrupt that the handler is to be installed for.

The following symbols are predefined:

Predefined Symbol	Meaning
<code>SRTK_INT_TIMER_A</code>	Timer A interrupt
<code>SRTK_INT_TIMER_B</code>	Timer B interrupt

Predefined Symbol	Meaning
SRTK_INT_TIMER_D	Timer D interrupt
SRTK_INT_GL_0	Gigalink 0 interrupt
...	...
SRTK_INT_GL_3	Gigalink 3 interrupt
SRTK_INT_MACROTICK	Macrotick interrupt
SRTK_INT_FWD_TIMER_A	Forwarded Timer A interrupt
SRTK_INT_FWD_TIMER_B	Forwarded Timer B interrupt
SRTK_INT_IO_ETH	I/O Ethernet interrupt

Note

Level-triggered interruptshave to be acknowledged in the interrupt service routine before they are enabled globally again.

Handler Specifies the pointer to the interrupt service routine.

Return value

This function returns the address of the interrupt service routine that was previously installed for this interrupt.

Example

The Timer A interrupt is to call the function `timera_interrupt` (see also `srtk_start_isr_timerA`).

First write the interrupt service routine `timera_interrupt`:

```
void timera_interrupt(void)
{
...
}
```

Then install the interrupt vector at the beginning of your application:

```
srtk_set_interrupt_vector(
    SRTK_INT_TIMER_A,
    (Srtk_Int_Handler_Type) timera_interrupt);
```

Related topics**References**

RTLIB_INT_ENABLE.....	106
Timer Interrupt Control.....	82

RTLIB_INT_DISABLE

Syntax	<code>RTLIB_INT_DISABLE()</code>						
Include file	<code>SrtkStd.h</code>						
Purpose	To globally disable the interrupts.						
	Note Use this macro only in conjunction with <code>RTLIB_INT_ENABLE</code> .						
Return value	None						
Related topics	References						
	<table><tr><td><code>RTLIB_INT_ENABLE</code>.....</td><td>106</td></tr><tr><td><code>srtk_disable_hardware_int</code>.....</td><td>94</td></tr><tr><td><code>srtk_disable_hardware_int_bm</code>.....</td><td>95</td></tr></table>	<code>RTLIB_INT_ENABLE</code>	106	<code>srtk_disable_hardware_int</code>	94	<code>srtk_disable_hardware_int_bm</code>	95
<code>RTLIB_INT_ENABLE</code>	106						
<code>srtk_disable_hardware_int</code>	94						
<code>srtk_disable_hardware_int_bm</code>	95						

RTLIB_INT_ENABLE

Syntax	<code>RTLIB_INT_ENABLE()</code>
Include file	<code>SrtkStd.h</code>
Purpose	To globally enable the interrupts.
Description	The only hardware interrupts that are available are the ones that are also enabled by <code>srtk_enable_hardware_int</code> .
	Note Use this macro only in conjunction with <code>RTLIB_INT_DISABLE</code> .

Return value None

Related topics References

RTLIB_INT_DISABLE.....	106
srtk_enable_hardware_int.....	96
srtk_enable_hardware_int_bm.....	97

RTLIB_INT_RESTORE

Syntax `void RTLIB_INT_RESTORE(UInt32 var_name)`

Include file SrtkStd.h

Purpose To restore the previous interrupt state after calling `RTLIB_INT_SAVE_AND_DISABLE`.

Note

Use this macro only in conjunction with `RTLIB_INT_SAVE_AND_DISABLE`.

Parameters `var_name` Returns the value of the previously executed macro `RTLIB_INT_SAVE_AND_DISABLE`.

Return value None

RTLIB_INT_SAVE_AND_DISABLE

Syntax `RTLIB_INT_SAVE_AND_DISABLE(UInt32 var_name)`

Include file SrtkStd.h

Purpose	To save the current interrupt status and globally disable the interrupts.
----------------	---

Note

Use this macro only in conjunction with **RTLIB_INT_RESTORE**.

Parameters	var_name Specifies the variable to store the interrupt status.
-------------------	---

Return value	None
---------------------	------

Example

```
void restore(void)
{
    UInt32 msr_state;

    RTLIB_INT_SAVE_AND_DISABLE(msr_state);
    /* Save the value of the EE bit in MSR and disable interrupts*/
    ...
    RTLIB_INT_RESTORE(msr_state);
    /* Restore the EE bit in MSR at the end of the function*/
}
```

Related topics**References**

RTLIB_INT_RESTORE	107
---	-----

RTLIB_SRT_DISABLE

Syntax

RTLIB_SRT_DISABLE()

Include file

SrtkStd.h

Purpose

To disable the hardware interrupt for the sampling rate timer when the interrupts are still globally enabled (see **RTLIB_INT_ENABLE**).

Description

Timer A is used as the sampling rate timer for MicroLabBox. This function sets the corresponding bit of the Interrupt Mask Register (IMR).

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

RTLIB_INT_ENABLE.....	106
RTLIB_SRT_ENABLE.....	109

RTLIB_SRT_ENABLE

Syntax	<code>RTLIB_SRT_ENABLE()</code>
---------------	---------------------------------

Include file	<code>SrtkStd.h</code>
---------------------	------------------------

Purpose	To enable the hardware interrupt for the sampling rate timer.
----------------	---

Description	Timer A is used as the sampling rate timer for MicroLabBox. This function only clears the corresponding bit of the Interrupt Mask Register (IMR). However, the hardware interrupt for Timer A is available only when the interrupts are globally enabled (see RTLIB_INT_ENABLE).
--------------------	--

Return value	None
---------------------	------

Related topics	References
-----------------------	------------

RTLIB_INT_ENABLE.....	106
RTLIB_SRT_DISABLE.....	108

Subinterrupt Handling

Introduction	Subinterrupt handling provides functions to extend one hardware interrupt to multiple software subinterrupts.
---------------------	---

Where to go from here	Information in this section
	Basic Principles of Subinterrupt Handling 111 Provides information on the subinterrupt handling principles.
	Example of Using a Subinterrupt Sender 111 Gives you instructions on implementing a subinterrupt sender.
	Example of Using a Subinterrupt Handler 112 Gives you instructions on implementing a subinterrupt handler.
	Example of Using a Subinterrupt Receiver 113 Gives you instructions on implementing a subinterrupt receiver.
	Data Types for Subinterrupt Handling 115 Provides the definition of the data types used by the subinterrupt module.
	dssint_define_int_sender 116 To define an interrupt sender.
	dssint_define_int_sender_1 118 To define an interrupt sender.
	dssint_define_int_receiver 120 To define an interrupt receiver.
	dssint_define_int_receiver_1 122 To define an interrupt receiver.
	dssint_subint_disable 124 To disable subinterrupts.
	dssint_subint_enable 125 To enable subinterrupts.
	dssint_interrupt 126 To trigger a subinterrupt.
	dssint_decode 126 To find out which subinterrupts are pending.
	dssint_acknowledge 127 To acknowledge pending subinterrupts.
	dssint_subint_reset 128 To clear pending subinterrupts.

Basic Principles of Subinterrupt Handling

Introduction

In dSPACE multiprocessor systems, interrupts can be dispatched between processors. Typically, there is only one hardware line between processors. To allow multiple different interrupt signals to be sent from a sender to a receiver, subinterrupt handling is provided which introduces logical interrupt sources. The subinterrupt handling meets the following goals:

- To trigger and handle multiple subinterrupts using a single hardware interrupt line.
- To allow that multiple different subinterrupts are pending at the receiver.
- To transmit and dispatch interrupts between several processors.
- To define interrupt senders/receivers to transmit subinterrupts.
- To use multiple senders and receivers at one processor.
- To get a point-to-point interrupt connection between two processors using a combination of sender and receiver.
- To make priority-based interrupt arbitration available (optional).
- Subinterrupts stay pending if they are disabled at the moment they occur.

Method

The following steps are necessary to program a subinterrupt handling between two applications:

- 1 Install a subinterrupt sender in your application that sends an interrupt.
- 2 Write an interrupt handler in your application that receives the interrupt.
- 3 Install a subinterrupt receiver in your application that receives the interrupt.

Example

See the following examples for more information:

- [Example of Using a Subinterrupt Sender on page 111](#)
- [Example of Using a Subinterrupt Handler on page 112](#)
- [Example of Using a Subinterrupt Receiver on page 113](#)

Example of Using a Subinterrupt Sender

Example

The following example shows the source code for the interrupt sender. It is defined for 16 subinterrupts. Every time the background loop is interrupted by timer 0, the subinterrupt 3 is sent to the receiver. The dual-port memory width is 16 bit and the accesses are direct.

```
#include <Brtenv.h>
#include <Defxxxx.h>      /* xxxx stands for the dSPACE */
#include <Mydefs.h>        /* board, e.g., 1401 for DS1401 */
dssint_sender_type *sender;
```

```

void isr_t0()
{
    dssint_interrupt(sender, 3);
}
void main()
{
    sender = dssint_define_int_sender_1(
        16,                                /* number of subinterrupts*/
        SUBINT_ADDR,                      /* start address of int. info */
        ACK_ADDR,                         /* start address of ack. info */
        SENDER_ADDR,                      /* trigger address */
        DPM_TARGET_DIRECT,                /* e.g., PHS bus base address */
        16,                                /* dual-port memory width */
        DPM_ACCESS_DIRECT,                /* pointer to write function */
        DPM_ACCESS_DIRECT);               /* pointer to read function */
    /* ... initialize timer 0 ... */
    global_enable();
    while(1);
}

```

Related topics**Basics**

Basic Principles of Subinterrupt Handling	111
---	-----

Examples

Example of Using a Subinterrupt Handler	112
Example of Using a Subinterrupt Receiver	113

Example of Using a Subinterrupt Handler

Example

The example shows an interrupt handler for the dSPACE real-time kernel.

When the interrupt is triggered, the processor dispatches it to `my_handler`, where it is acknowledged by calling `dssint_acknowledge`. The function `dssint_decode` is called repetitively and returns the according subinterrupt number for every pending subinterrupt. For every subinterrupt, one task is registered by calling `rtk_register_task`.

`rtk_register_task` sets the task state for the according task to 'ready' when the task priority is not the highest of all registered tasks. The function internally stores the task registered with the highest priority and returns a pointer to it. `rtk_register_task` does not schedule tasks.

Once all tasks are registered, the "task" pointer holds the one with the highest priority. This task can be of a lower, equal or higher priority than the currently running task. Via the "task" pointer the scheduler is called – this is the reason

why the state of the task registered with the highest priority must not be set to 'ready'.

The scheduler clears the stored information about the task registered with the highest priority.

```
void my_handler()
{
    rtk_p_task_control_block task = 0;
    int sub_int;
    dssint_acknowledge(receiver); /* interrupt acknowledge */
    /* Register tasks */
    do {
        if ( (sub_int = dssint_decode(receiver)) >= 0)
            task = rtk_register_task(S_MYSERVICE, sub_int);
    } while(subint >= 0);
    /* Call the scheduler */
    if (task)
        rtk_scheduler(task);
}
```

Related topics

Basics

Basic Principles of Subinterrupt Handling.....	111
--	-----

Examples

Example of Using a Subinterrupt Receiver.....	113
Example of Using a Subinterrupt Sender.....	111

Example of Using a Subinterrupt Receiver

Example

In this example, a receiver with 16 subinterrupts is defined. It is assumed that the kernel installs the function `my_handler` (refer to the [Example of Using a Subinterrupt Handler](#) on page 112) as an interrupt service routine for subinterrupts. The `main` function enables interrupts and enters the background task after creating and binding the tasks to the subinterrupts.

```
#include <Brtenv.h>
#include <Defxxxx.h>      /* xxxx stands for the dSPACE */
                           /* board, e.g. 1401 for DS1401 */
void slave0_task(void)
{
    /*...*/
};
dssint_receiver_type receiver;
```

```
void main()
{
    rtk_p_task_control_block task;
    receiver = dssint_define_int_receiver_1(
        16,                                /* number of subinterrupts*/
        SUBINT_ADDR,                      /* start address of int. info */
        ACK_ADDR,                          /* start address of ack. info */
        RECEIVER_ADDR,                    /* receiver address */
        DPM_TARGET_DIRECT,                /* e.g. PHS bus base address */
        16,                                /* dual-port memory width */
        DPM_ACCESS_DIRECT,                /* pointer to write function */
        DPM_ACCESS_DIRECT);               /* pointer to read function */
    /* ... */
    task = rtk_create_task((rtk_task_fcn_type)slave0_task, 1,
                           ovc_queue, rtk_default_overrun_fcn, 10,0);
    rtk_bind_interrupt(S_SLAVE, 0, task, 0.0, C_LOCAL, 0, 0);
    /*...*/
    global_enable();
    while(1);
}
```

Related topics

Basics

Basic Principles of Subinterrupt Handling.....	111
--	-----

Examples

Example of Using a Subinterrupt Handler.....	112
Example of Using a Subinterrupt Sender.....	111

Data Types for Subinterrupt Handling

dssint_sender_type

```
typedef struct{
    unsigned int    nr_sint;      /* number of subinterrupts */
    unsigned long   sint_addr;   /* start address of the */
                                /* interrupt info */
    unsigned long   ack_addr;   /* start address of the */
                                /* acknowledge info */
    unsigned long   sender_addr; /* writing to this address */
                                /* triggers interrupt */
    unsigned int    nr_words;    /* number of words */
                                /* needed for nr_sint */
    unsigned long*  request;    /* pointer to local copy */
                                /* of sint_addr */
    long           target;     /* e.g. PHS bus base address */
    unsigned int    sint_mem_width;
                                /* width of the*/
                                /* dual-port memory */
    dpm_write_fcn_t write_fcn; /* pointer to write function */
    dpm_read_fcn_t  read_fcn;   /* pointer to read function */
    unsigned int    sint_mem_shift;
                                /* internal performance */
                                /* improvement */
}dssint_sender_type;
```

dssint_receiver_type

```
typedef struct{
    unsigned int    nr_sint;      /* number of subinterrupts */
    unsigned long   sint_addr;   /* start address of the */
                                /* interrupt info */
    unsigned long   ack_addr;   /* start address of the */
                                /* acknowledge info */
    unsigned long   receiver_addr;
                                /* reading from this address */
                                /* performs hardware ack of */
                                /* interrupt */
    unsigned int    nr_words;    /* number of words */
                                /* needed for nr_sint */
    unsigned long*  acknowledge;
                                /* pointer to local copy */
                                /* of ack_addr */
    unsigned long*  state;      /* pointer to state info */
    long           target;     /* e.g. PHS bus base address */
    unsigned int    sint_mem_width;
                                /* width of the*/
                                /* dual-port memory */
    unsigned int    state_position;
                                /* decode position in state */
    dpm_write_fcn_t write_fcn; /* pointer to write function */
    dpm_read_fcn_t  read_fcn;   /* pointer to read function */
    unsigned int    sint_mem_shift;
                                /* internal performance */
                                /* improvement */
    unsigned long*  enable_flag; /* for pending interrupts */
    dssint_ack_fcn_t ack_fcn;   /* pointer to interrupt acknowledge function */
}dssint_receiver_type;
```

Related topics	Basics
	Basic Principles of Subinterrupt Handling 111
Examples	
	Example of Using a Subinterrupt Handler 112 Example of Using a Subinterrupt Receiver 113 Example of Using a Subinterrupt Sender 111

dssint_define_int_sender

Syntax	<pre>dssint_sender_type* dssint_define_int_sender(unsigned int nr_subinterrupts, unsigned long subint_addr, unsigned long ack_addr, unsigned long sender_addr, long target, unsigned int sint_mem_width, dpm_write_fcn_t write_fcn, dpm_read_fcn_t read_fcn)</pre>
Include file	dssint.h
Purpose	To define the sender of a subinterrupt.
Description	<p>The function defines an interrupt sender and returns a handle to it. A sender processor can have multiple receiver processors to pass interrupts to. The handle identifies where to send an interrupt. The function initializes all memory locations in the dual-port memory used for the subinterrupt handling with 0.</p> <p>The functions <code>dssint_define_int_sender</code> and <code>dssint_define_int_receiver</code> define the sender and receiver of a subinterrupt in the following way:</p> <p>When subinterrupts are sent before the receiver is initialized, these interrupts are stored. After the receiver is initialized these interrupts are passed to the receiver and processed.</p>

Note

- The behavior described above can cause overflows. To avoid this, use the functions `dssint_define_int_sender_1` and `dssint_define_int_receiver_1` instead.
- If you define a sender of a subinterrupt via the function `dssint_define_int_sender`, you must define the receiver via the function `dssint_define_int_receiver`.

Parameters

nr_subinterrupts Specifies the number of different subinterrupts to be transferred. This is necessary to define the width of the memory portion which passes the subinterrupt information. The number of subinterrupts must be equal for sender and receiver.

subint_addr Specifies the memory location the subinterrupt information is passed to.

ack_addr Specifies the memory location the acknowledgment information from the receiver is passed to.

sender_addr Specifies the pointers to the memory location that triggers the interrupt by writing to it (hardware trigger). This address can be the same as `subint_addr`.

target Specifies the address of the target memory, for example, a PHS bus address or COM port number. This parameter is meaningless for direct access.

sint_mem_width Specifies the width of the dual-port memory.

write_fcn Specifies the address of a function that performs a write access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

read_fcn Specifies the address of a function that performs a read access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

Return value

This function returns the handle to an interrupt sender. The function returns 0 if an error occurred.

Example

See [Example of Using a Subinterrupt Sender](#) on page 111.

Related topics**Basics**

[Basic Principles of Subinterrupt Handling](#)..... 111

Examples

[Example of Using a Subinterrupt Sender](#)..... 111

References

dssint_define_int_receiver	120
dssint_define_int_receiver_1	122
dssint_define_int_sender_1	118

dssint_define_int_sender_1

Syntax

```
dssint_sender_type* dssint_define_int_sender_1(
    unsigned int nr_subinterrupts,
    unsigned long subint_addr,
    unsigned long ack_addr,
    unsigned long sender_addr,
    long target,
    unsigned int sint_mem_width,
    dpm_write_fcn_t write_fcn,
    dpm_read_fcn_t read_fcn)
```

Include file**dssint.h****Purpose**

To define the sender of a subinterrupt.

Description

The function defines an interrupt sender and returns a handle to it. A sender processor can have multiple receiver processors to pass interrupts to. The handle identifies where to send an interrupt. The function initializes all memory locations in the dual-port memory used for the subinterrupt handling with 0.

The functions `dssint_define_int_sender_1` and `dssint_define_int_receiver_1` define the sender and receiver of a subinterrupt in the following way:

When subinterrupts are sent before the receiver is initialized, these interrupts are not stored to avoid overflows.

Note

If you define a sender of a subinterrupt via the function `dssint_define_int_sender_1`, you have to define the receiver via the function `dssint_define_int_receiver_1`.

Parameters

nr_subinterrupts Specifies the number of different subinterrupts to be transferred. The number of subinterrupts must be equal for sender and receiver. See [dssint_define_int_sender](#) on page 116.

subint_addr Specifies the memory location the subinterrupt information is passed to.

ack_addr Specifies the memory location the acknowledgment information from the receiver is passed to.

sender_addr Specifies the pointers to the memory location that triggers the interrupt by writing to it (hardware trigger). This address can be the same as `subint_addr`.

target Specifies the address of the target memory, for example, a PHS bus address or COM port number. This parameter is meaningless for direct access.

sint_mem_width Specifies the width of the dual-port memory.

write_fcn Specifies the address of a function that performs a write access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

read_fcn Specifies the address of a function that performs a read access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

Return value

This function returns the handle to an interrupt sender. The function returns 0 if an error occurred.

Example

See [Example of Using a Subinterrupt Sender](#) on page 111.

Related topics**Basics**

[Basic Principles of Subinterrupt Handling](#)..... 111

Examples

[Example of Using a Subinterrupt Sender](#)..... 111

References

[dssint_define_int_receiver_1](#)..... 122

[dssint_define_int_sender](#)..... 116

dssint_define_int_receiver

Syntax

```
dssint_receiver_type *dssint_define_int_receiver(
    unsigned int nr_subinterrupts,
    unsigned long subint_addr,
    unsigned long ack_addr,
    unsigned long receiver_addr,
    long target,
    unsigned int sint_mem_width,
    dpm_write_fcn_t write_fcn,
    dpm_read_fcn_t read_fcn)
```

Include file

dssint.h

Purpose

To define the receiver of a subinterrupt.

Description

The function reads from the `receiver_addr` to enable interrupt triggering by the sender. It defines an interrupt receiver and returns a handle to it. A receiver processor can have multiple sender processors from which interrupts are retrieved. The handle identifies the appropriate subinterrupt vector and receiving information table for a specific sender.

The functions `dssint_define_int_receiver` and `dssint_define_int_sender` define the receiver and sender of a subinterrupt in the following way:

When subinterrupts are sent before the receiver is initialized, these interrupts are stored. After the receiver is initialized, these interrupts are passed to the receiver and processed.

Note

- The behavior described above can cause overflows. To avoid this, use the functions `dssint_define_int_sender_1` and `dssint_define_int_receiver_1` instead.
- If you define a receiver of a subinterrupt via the function `dssint_define_int_receiver`, you have to define the sender via the function `dssint_define_int_sender`.

Parameters

nr_subinterrupts Specifies the number of different subinterrupts to be transferred. The number of subinterrupts must be equal for sender and receiver. See [dssint_define_int_sender](#) on page 116.

subint_addr Specifies the memory location the subinterrupt information is passed to.

ack_addr Specifies the memory location the acknowledgment information from the receiver is passed to.

receiver_addr Specifies the pointers to the memory location that acknowledges the interrupt by reading it (hardware acknowledge).

target Specifies the address of the target memory, for example, a PHS bus address or COM port number. This parameter is meaningless for direct access.

sint_mem_width Specifies the width of the dual-port memory.

write_fcn Specifies the address of a function that performs a write access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

read_fcn Specifies the address of a function that performs a read access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

Return value

This function returns the address of an interrupt receiver. The function returns 0 if an error occurred.

Example

See [Example of Using a Subinterrupt Receiver](#) on page 113.

Related topics	Basics
	Basic Principles of Subinterrupt Handling..... 111
	Examples
	Example of Using a Subinterrupt Receiver..... 113
	References
	dssint_define_int_receiver_1..... 122 dssint_define_int_sender..... 116 dssint_define_int_sender_1..... 118

dssint_define_int_receiver_1

Syntax	<pre>dssint_receiver_type *dssint_define_int_receiver_1(unsigned int nr_subinterrupts, unsigned long subint_addr, unsigned long ack_addr, unsigned long receiver_addr, long target, unsigned int sint_mem_width, dpm_write_fcn_t write_fcn, dpm_read_fcn_t read_fcn)</pre>
Include file	dssint.h
Purpose	To define the receiver of a subinterrupt.
Description	<p>The function reads from the <code>receiver_addr</code> to enable interrupt triggering by the sender. It defines an interrupt receiver and returns a handle to it. A receiver processor can have multiple sender processors from which interrupts are retrieved. The handle identifies the appropriate subinterrupt vector and receiving information table for a specific sender.</p> <p>The functions <code>dssint_define_int_receiver_1</code> and <code>dssint_define_int_sender_1</code> define the receiver and sender of a subinterrupt in the following way:</p>

When subinterrupts are sent before the receiver is initialized, these interrupts will not be stored to avoid overflows.

Note

If you define a receiver of a subinterrupt via the function `dssint_define_int_receiver_1`, you must define the sender via the function `dssint_define_int_sender_1`.

Parameters

nr_subinterrupts Specifies the number of different subinterrupts to be transferred. The number of subinterrupts must be equal for sender and receiver. See [dssint_define_int_sender](#) on page 116.

subint_addr Specifies the memory location the subinterrupt information is passed to.

ack_addr Specifies the memory location the acknowledgment information from the receiver is passed to.

receiver_addr Specifies the pointers to the memory location that acknowledges the interrupt by reading it (hardware acknowledge).

target Specifies the address of the target memory, for example, a PHS bus address or COM port number. This parameter is meaningless for direct access.

sint_mem_width Specifies the width of the dual-port memory.

write_fcn Specifies the address of a function that performs a write access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

read_fcn Specifies the address of a function that performs a read access to the dual-port memory. Set this parameter to 0 for direct access (if all supported memories that are accessed directly have the same width).

Return value

This function returns the address of an interrupt receiver. The function returns 0 if an error occurred.

Example

See [Example of Using a Subinterrupt Receiver](#) on page 113.

Related topics

Basics

Basic Principles of Subinterrupt Handling..... 111

Examples

Example of Using a Subinterrupt Receiver..... 113

References

dssint_define_int_receiver..... 120

dssint_define_int_sender_1..... 118

dssint_subint_disable

Syntax

```
void dssint_subint_disable(  
    dssint_receiver_type *receiver,  
    unsigned int subinterrupt)
```

Include file

dssint.h

Purpose

To disable a subinterrupt.

Description

After initialization, all subinterrupts are enabled. You must disable the subinterrupt explicitly via this function.

Parameters

receiver Specifies the receiver handler the subinterrupt is located in.

subinterrupt Specifies the subinterrupt to reset.

Example

```
...  
dssint_subint_disable(my_receiver, 5);  
...
```

Related topics

Basics

Basic Principles of Subinterrupt Handling	111
---	-----

References

dssint_subint_enable	125
dssint_subint_reset	128

dssint_subint_enable

Syntax

```
void dssint_subint_enable(
    dssint_receiver_type *receiver,
    unsigned int subinterrupt)
```

Include file

dssint.h

Purpose

To enable a subinterrupt.

DescriptionAfter initialization, all subinterrupts are enabled. Use this function if you disabled a subinterrupt via `dssint_subint_disable` before.

Parameters**receiver** Specifies the receiver handler the subinterrupt is located in.**subinterrupt** Specifies the subinterrupt to reset.

Example

```
...
dssint_subint_enable(my_receiver, 5);
...
```

Related topics

Basics

Basic Principles of Subinterrupt Handling	111
---	-----

References

dssint_subint_disable	124
dssint_subint_reset	128

dssint_interrupt

Syntax

```
void dssint_interrupt(
    dssint_sender_type *sender,
    unsigned int sub_interrupt)
```

Include file

`dssint.h`

Purpose

To write the subinterrupt information to the specified memory location and to trigger the interrupt.

Parameters

sender Specifies the handle of the interrupt sender.
sub_interrupt Specifies the subinterrupt to be triggered. Values are within the range 0 ... nr_subinterrupts. Parameter nr_subinterrupts is defined by `dssint_define_int_sender` (or `dssint_define_int_sender_1`) and `dssint_define_int_receiver` (or `dssint_define_int_receiver_1`).

Example

See [Example of Using a Subinterrupt Sender](#) on page 111.

Related topics

Basics

Basic Principles of Subinterrupt Handling	111
---	-----

Examples

Example of Using a Subinterrupt Handler	112
---	-----

References

<code>dssint_define_int_receiver</code>	120
<code>dssint_define_int_receiver_1</code>	122
<code>dssint_define_int_sender</code>	116
<code>dssint_define_int_sender_1</code>	118

dssint_decode

Syntax

```
int dssint_decode(dssint_receiver_type *receiver)
```

Include file	dssint.h
Purpose	To identify the pending interrupts.
Description	This function is called repetitively within an interrupt handler. It processes the interrupt information of the receiver data structure that was given by dssint_acknowledge , determines the pending subinterrupt with the highest priority and returns it to the handler. The pending subinterrupt with the highest priority is the one with the smallest subinterrupt number.
Parameters	receiver Specifies the receiver handler the subinterrupt is located in.
Return value	This function returns the number of the pending subinterrupt with highest priority. If there is no pending subinterrupt left, the function returns SINT_NO_SUBINT ("−1").
Example	See Example of Using a Subinterrupt Handler on page 112.
Related topics	Basics Basic Principles of Subinterrupt Handling 111 Examples Example of Using a Subinterrupt Handler 112 References dssint_acknowledge 127

dssint_acknowledge

Syntax	<code>void dssint_acknowledge(dssint_receiver_type *receiver)</code>
Include file	dssint.h

Purpose	To acknowledge pending subinterrupts.										
Description	<p>This function acknowledges the interrupt by reading <code>receiver->receiver_addr</code> (hardware acknowledge), and copies the subinterrupt information to the receiver data structure. Then it performs the software acknowledgment for every pending subinterrupt.</p> <p>For information on the receiver data structure, refer to the type definition given in Data Types for Subinterrupt Handling on page 115.</p>										
Parameters	<code>receiver</code> Specifies the receiver handler the subinterrupt is located in.										
Example	See Example of Using a Subinterrupt Handler on page 112.										
Related topics	<p>Basics</p> <table border="0"> <tr> <td style="background-color: #e0e0e0;">Basic Principles of Subinterrupt Handling.....</td> <td style="background-color: #e0e0e0;">111</td> </tr> </table> <p>Examples</p> <table border="0"> <tr> <td style="background-color: #e0e0e0;">Example of Using a Subinterrupt Handler.....</td> <td style="background-color: #e0e0e0;">112</td> </tr> </table> <p>References</p> <table border="0"> <tr> <td style="background-color: #e0e0e0;">Data Types for Subinterrupt Handling.....</td> <td style="background-color: #e0e0e0;">115</td> </tr> <tr> <td style="background-color: #e0e0e0;">dssint_define_int_receiver.....</td> <td style="background-color: #e0e0e0;">120</td> </tr> <tr> <td style="background-color: #e0e0e0;">dssint_define_int_receiver_1.....</td> <td style="background-color: #e0e0e0;">122</td> </tr> </table>	Basic Principles of Subinterrupt Handling	111	Example of Using a Subinterrupt Handler	112	Data Types for Subinterrupt Handling	115	dssint_define_int_receiver	120	dssint_define_int_receiver_1	122
Basic Principles of Subinterrupt Handling	111										
Example of Using a Subinterrupt Handler	112										
Data Types for Subinterrupt Handling	115										
dssint_define_int_receiver	120										
dssint_define_int_receiver_1	122										

dssint_subint_reset

Syntax	<pre>void dssint_subint_reset(dssint_receiver_type *receiver, unsigned int subinterrupt)</pre>
Include file	<code>dssint.h</code>
Purpose	To clear a pending subinterrupt.

Parameters	receiver Specifies the receiver handler the subinterrupt is located in. subinterrupt Specifies the subinterrupt to reset.
-------------------	--

Example

```
...  
dssint_subint_reset(my_receiver, 5);  
...
```

Related topics**Basics**

Basic Principles of Subinterrupt Handling.....	111
--	-----

References

Data Types for Subinterrupt Handling.....	115
dssint_subint_disable.....	124
dssint_subint_enable.....	125

Message Handling

Purpose	To configure and generate messages.
Where to go from here	Information in this section
	<p>Basic Principles of Message Handling..... 131 Information on the Message module's basic principles.</p> <p>Data Types and Symbols for Message Handling..... 132 Information on the data types and symbols defined in the Message module.</p> <p>msg_error_set..... 134 To generate an error message.</p> <p>msg_warning_set..... 135 To generate a warning message.</p> <p>msg_info_set..... 135 To generate an information message.</p> <p>msg_set..... 136 To generate a message of the defined message class.</p> <p>msg_error_printf..... 138 To generate an error message with arguments using the <code>printf</code> format.</p> <p>msg_warning_printf..... 140 To generate a warning message with arguments using the <code>printf</code> format.</p> <p>msg_info_printf..... 141 To generate an information message with arguments using the <code>printf</code> format.</p> <p>msg_printf..... 142 To generate a message of the specified class with arguments using the <code>printf</code> format.</p> <p>msg_default_dialog_set..... 144 To specify the default dialog type for the selected message class.</p> <p>msg_mode_set..... 145 To set the mode of the message buffer.</p> <p>msg_reset..... 146 To reset the message buffer and clear the values of the last error.</p> <p>msg_last_error_number..... 146 To read the number of the last generated error message.</p> <p>msg_last_error_submodule..... 147 To read the submodule of the last generated error message.</p>

msg_error_clear	149
---------------------------------------	-----

To set the number of the last generated error to 0 and the submodule of the last generated error message to MSG_SM_NONE .

msg_error_hook_set	150
--	-----

To install a hook function.

msg_init	151
--------------------------------	-----

To initialize the message handling.

Basic Principles of Message Handling

Introduction

The Message module provides functions to generate error, warning, and information messages to be displayed by the dSPACE experiment software. Messages are generated by the processor board, written to a message buffer, and sent to the host PC. On the host PC, the dSPACE experiment software displays the messages in the log window and writes them to the log file. Each message consists of a message number and the message string. To use the message module, you have to initialize the board via the initialization function `init()`.

Message characteristics

There are two predefined symbols that define the message buffer. The symbol `MSG_STRING_LENGTH` specifies the maximum length of a generated message. If a message exceeds the given length, it is truncated. The symbol `MSG_BUFFER_LENGTH` specifies the maximum number of messages that can be stored to the reserved memory. The behavior of the message buffer is controlled by the `msg_mode_set` function. The values of the message and buffer lengths are defined in `MsgXXXX.h` (XXXX denotes the relevant dSPACE board) or `StrkMsg.h` when you use DS1007 or MicroLabBox.

For MicroLabBox, the following values are used:

Predefined Symbol	Default Value
<code>MSG_STRING_LENGTH</code>	480 characters
<code>MSG_BUFFER_LENGTH</code>	256 messages

Message types

There are four message types:

Type	Representation in the dSPACE Experiment Software
ERROR	Dialog box containing the message text and entry in the Log window beginning with ERROR
WARNING	Entry in the Log window beginning with WARNING

Type	Representation in the dSPACE Experiment Software
INFO	Entry in the Log window
LOG	Entry in the Log file only

The following table gives examples for the three message types ERROR, WARNING, and INFO:

Module	Message Type	Board Name	Submodule	Message Text
Platform:	ERROR			Board is not present or expansion box is off.
DataKernel:	WARNING			Data connection not valid!
Real-Time Processor:		#1 DS1202 -	RTLib:	System started. (0)

Data Types and Symbols for Message Handling

Data types

The following data types are defined:

msg_string_type

```
typedef char msg_string_type;
```

msg_no_type

```
typedef Int32 msg_no_type;
```

msg_class_type

```
typedef enum msg_class_type;
```

msg_dialog_type

```
typedef enum msg_dialog_type;
```

msg_submodule_type

```
typedef UInt32 msg_submodule_type;
```

msg_hookfcn_type

```
typedef int (*msg_hookfcn_type)(msg_submodule_type, msg_no_type);
```

The following symbols are defined:

Predefined Symbol	Message refers to ...
MSG_SM_NONE	No specific module (default)
MSG_SM_USER	User messages
MSG_SM_CAN1401	RTLib: CAN (DS1401)
MSG_SM_CAN2202	RTLib: CAN (DS2202)
MSG_SM_CAN2210	RTLib: CAN (DS2210)

Predefined Symbol	Message refers to ...
MSG_SM_CAN2211	RTLib: CAN (DS2211)
MSG_SM_CAN4302	RTLib: CAN (DS4302)
MSG_SM_DIO1401	RTLib: Digital I/O (DS1401)
MSG_SM_DS1104SLVLIB	RTLib: Slave DSP (DS1104)
MSG_SM_DS4501	RTLib: DS4501 functions
MSG_SM_DS4502	RTLib: DS4502 functions
MSG_SM_DSBYPASS	RTI: Bypass Blockset
MSG_SM_DSCAN	RTLib: CAN support
MSG_SM_DSETH	RTI: RTI Ethernet Blockset
MSG_SM_DSFR	RTLib: FlexRay support
MSG_SM_DSJ1939	J1939 Support in RTI CAN MultiMessage Blockset
MSG_SM_DSSER	RTLib: Serial interface
MSG_SM_ECU_POD	ECU PODs (DS5xx)
MSG_SM_ECU1401	RTLib: ECU interface (DS1401)
MSG_SM_HOSTSERV	Host services
MSG_SM_LIN	RTLib: LIN support
MSG_SM_REALMOTION	RealMotion / MotionDesk
MSG_SM_RTI	Real-Time Interface
MSG_SM_RTICAN	RTI: CAN Blockset
MSG_SM_RTICAN1401	RTI: CAN Blockset (DS1401)
MSG_SM_RTICAN2202	RTI: CAN Blockset (DS2202)
MSG_SM_RTICAN2210	RTI: CAN Blockset (DS2210)
MSG_SM_RTICAN2211	RTI: CAN Blockset (DS2211)
MSG_SM_RTICAN4302	RTI: CAN Blockset (DS4302)
MSG_SM_RTICANMM	RTI: CAN MultiMessage Blockset
MSG_SM_RTIFLEXRAY	RTI: FlexRay Blockset
MSG_SM_RTIFLEXRAYCONFIG	RTI: FlexRay Configuration Blockset
MSG_SM_RTILINMM	RTI: LIN MultiMessage Blockset
MSG_SM_RTIMP	RTI-MP (Real-Time Interface for multiprocessor systems)
MSG_SM_RTKERNEL	Real-Time Kernel
MSG_SM_RTLIB	Real-Time Board Library
MSG_SM_RTOSAL	RTOS Abstractionlayer
MSG_SM RTPYTHON	RTPython interpreter
MSG_SM_SIMENG	RTI: Simulation engine

msg_error_set

Syntax

```
void msg_error_set(  
    msg_submodule_type module,  
    msg_no_type msg_no,  
    msg_string_type *msg)
```

Include file

dsmsg.h

Purpose

To generate an error message.

Note

If there is a hook function installed (see [msg_error_hook_set](#)), the hook function is called before the error message is generated.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 132.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

msg Specifies the message string (for information on the maximum length, see [Message characteristics](#) on page 131).

Return value

None

Related topics

Basics

[Basic Principles of Message Handling](#)..... 131

References

[msg_error_hook_set](#)..... 150
[msg_error_printf](#)..... 138

msg_warning_set

Syntax

```
void msg_warning_set(
    msg_submodule_type module,
    msg_no_type msg_no,
    msg_string_type *msg)
```

Include file

`dsmsg.h`

Purpose

To generate a warning message.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 132.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

msg Specifies the message string (for information on the maximum length, see [Message characteristics](#) on page 131).

Return value

None

Related topics

Basics

Basic Principles of Message Handling	131
--	-----

References

msg_warning_printf	140
--	-----

msg_info_set

Syntax

```
void msg_info_set(
    msg_submodule_type module,
    msg_no_type msg_no,
    msg_string_type *msg)
```

Include file	<code>dsmsg.h</code>				
Purpose	To generate an information message.				
Parameters	<p>module Specifies the predefined symbol of the application module generating the message. Use the module type <code>MSG_SM_USER</code> only for handcoded programs. For a list of all predefined symbols, refer to Data Types and Symbols for Message Handling on page 132.</p> <p>msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.</p> <p>msg Specifies the message string (for information on the maximum length, see Message characteristics on page 131).</p>				
Return value	None				
Related topics	<p>Basics</p> <table> <tr> <td>Basic Principles of Message Handling.....</td> <td>131</td> </tr> </table> <p>References</p> <table> <tr> <td>msg_info_printf.....</td> <td>141</td> </tr> </table>	Basic Principles of Message Handling	131	msg_info_printf	141
Basic Principles of Message Handling	131				
msg_info_printf	141				

msg_set

Syntax	<pre>void msg_set(msg_class_type msg_class, msg_dialog_type msg_dialog, msg_submodule_type module, msg_no_type msg_no, msg_string_type *msg)</pre>
Include file	<code>dsmsg.h</code>
Purpose	To generate a message of the defined message class.

Description	This function issues an error, information, or warning message that is displayed by the dSPACE experiment software, or a message that only appears in the log file. In addition to the other <code>msg_xxx_set</code> functions, the user can adjust the type of the message dialogs.
--------------------	---

Parameters	msg_class Specifies the type of the message. The following symbols are predefined:
-------------------	---

Predefined Symbol	Meaning
<code>MSG_MC_ERROR</code>	Error message
<code>MSG_MC_INFO</code>	Information message
<code>MSG_MC_WARNING</code>	Warning message
<code>MSG_MC_LOG</code>	Message appears only in the log file

msg_dialog	Specifies the type of the dialog. The following types are predefined:
-------------------	---

Predefined Symbol	Meaning
<code>MSG_DLG_NONE</code>	No dialog, silent mode
<code>MSG_DLG_OKCANCEL</code>	OK/Cancel dialog
<code>MSG_DLG_DEFAULT</code>	Dialog type specified by <code>msg_default_dialog_set</code>

Note

If you use a MicroLabBox, displaying messages in a dialog in ControlDesk is not supported. This parameter is not used and only provided for backward compatibility. All messages are displayed in the standard log.

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for hardcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 132.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

msg Specifies the message string (for information on the maximum length, see [Message characteristics](#) on page 131).

Return value	None
---------------------	------

Example

The following example issues an error message without a dialog.

```
msg_set(
    MSG_MC_ERROR,
    MSG_DLG_NONE,
    MSG_SM_USER,
    1,
    "This is an error message.");
```

Related topics**Basics**

[Basic Principles of Message Handling](#)..... 131

References

[msg_printf](#)..... 142

msg_error_printf

Syntax

```
int msg_error_printf(
    msg_submodule_type module,
    msg_no_type msg_no,
    char *format,
    arg1, arg2, etc.)
```

Include file

dsmsg.h

Purpose

To generate an error message with arguments using the `printf` format (see a standard C documentation).

Result

`printf` builds the message string with the standard C command arguments of `printf(char *format, arg1, arg2, etc.).` The string is then automatically given to `msg_error_set` to generate the message.

Note

If there is a hook function installed (see [msg_error_hook_set](#) on page 150), the hook function is called before the error message is generated.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for hardcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 132.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

format Specifies the string using the `printf` format.

arg1, arg2, etc. Specifies the optional arguments for the format string (see a standard C documentation).

Note

The length of the format string is not restricted, but the default value of the maximum length is specified by `MSG_STRING_LENGTH`, see [Message characteristics](#) on page 131. Longer messages are truncated.

Return value

This function returns the number of characters which were printed to the message buffer.

Example

This example shows how to generate an error message with the `printf` format:

```
#include <Brtenv.h>
/* An example integer value */
int num = 13;
void main()
{
    /* Initialization of the board */
    init();
    /* Write an error message to the message buffer using the printf format */
    msg_error_printf(MSG_SM_USER, 1, "The value of num is %i", num);
}
```

Related topics**Basics**

Basic Principles of Message Handling.....	131
---	-----

References

msg_error_hook_set.....	150
msg_error_set.....	134

msg_warning_printf

Syntax

```
int msg_warning_printf(
    msg_submodule_type module,
    msg_no_type msg_no,
    char *format,
    arg1, arg2, etc.)
```

Include file

dsmsg.h

Purpose

To generate a warning message with arguments using the `printf` format (see a standard C documentation).

Result

`printf` builds the message string with the standard C command arguments of `printf(char *format, arg1, arg2, etc.).` The string is then automatically passed to `msg_warning_set` to generate the message.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 132.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

format Specifies the string using the `printf` format.

arg1, arg2, etc. Specifies the optional arguments for the format string (see a standard C documentation).

Note

The length of the format string is not restricted, but the default value of the maximum length is specified by `MSG_STRING_LENGTH`, see [Message characteristics](#) on page 131. Longer messages are truncated.

Return value

This function returns the number of characters which were printed to the message buffer.

Related topics

Basics

[Basic Principles of Message Handling](#)..... 131

References

[msg_warning_set](#)..... 135

msg_info_printf

Syntax

```
int msg_info_printf(  
    msg_submodule_type module,  
    msg_no_type msg_no,  
    char *format,  
    arg1, arg2, etc.)
```

Include file

dsmsg.h

Purpose

To generate an information message with arguments using the `printf` format (see a standard C documentation).

Result

`printf` builds the message string with the standard C command arguments of `printf(char *format, arg1, arg2, etc.).` The string is then automatically given to `msg_info_set` to generate the message.

Parameters

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 132.

msg_no Specifies the number of the message within the range of -2³¹ ... 2³¹-1 defined by the user.

format Specifies the string using the `printf` format.

arg1, arg2, etc. Specifies the optional arguments for the format string (see a standard C documentation).

Note

The length of the format string is not restricted, but the default value of the maximum length is specified by `MSG_STRING_LENGTH`, see [Message characteristics](#) on page 131. Longer messages are truncated.

Return value	This function returns the number of characters which were printed to the message buffer.
---------------------	--

Related topics

Basics

[Basic Principles of Message Handling](#)..... 131

References

[msg_info_set](#)..... 135

msg_printf

Syntax

```
int msg_printf(
    msg_class_type msg_class,
    msg_dialog_type msg_dialog,
    msg_submodule_type module,
    msg_no_type msg_no,
    char *format,
    arg1, arg2, etc.)
```

Include file

`dsmsg.h`

Purpose

To generate a message of the specified class with arguments using the `printf` format (see a standard C documentation).

Result

`printf` builds the message string with the standard C command arguments of `printf(char *format, arg1, arg2, etc.).` The string is then automatically given to `msg_set` to generate the message.

Parameters

msg_class Specifies the type of the message. The following symbols are predefined:

Predefined Symbol	Meaning
MSG_MC_ERROR	Error message
MSG_MC_INFO	Information message
MSG_MC_WARNING	Warning message
MSG_MC_LOG	Message appears only in the log file

msg_dialog Specifies the type of the dialog. The following types are predefined:

Predefined Symbol	Meaning
MSG_DLG_NONE	No dialog, silent mode
MSG_DLG_OKCANCEL	OK/Cancel dialog
MSG_DLG_DEFAULT	Dialog type specified by <code>msg_default_dialog_set</code>

Note

If you use a MicroLabBox, displaying messages in a dialog in ControlDesk is not supported. This parameter is not used and only provided for backward compatibility. All messages are displayed in the standard log.

module Specifies the predefined symbol of the application module generating the message. Use the module type `MSG_SM_USER` only for handcoded programs. For a list of all predefined symbols, refer to [Data Types and Symbols for Message Handling](#) on page 132.

msg_no Specifies the number of the message within the range of $-2^{31} \dots 2^{31}-1$ defined by the user.

format Specifies the string using the `printf` format.

arg1, arg2, etc. Specifies the optional arguments for the format string (see a standard C documentation).

Note

The length of the format string is not restricted, but the default value of the maximum length is specified by `MSG_STRING_LENGTH`, see [Message characteristics](#) on page 131. Longer messages are truncated.

Return value	This function returns the number of characters which were printed to the message buffer.
---------------------	--

Related topics	Basics
	Basic Principles of Message Handling 131
	References
	msg_set 136

msg_default_dialog_set

Syntax	<pre>void msg_default_dialog_set(msg_class_type msg_class, msg_dialog_type msg_dialog)</pre>
---------------	---

Include file	dsmsg.h
---------------------	---------

Purpose	To specify the default dialog type for the selected message class.
----------------	--

Note

If you use a MicroLabBox, displaying messages in a dialog in ControlDesk is not supported. This function is not used and only provided for backward compatibility. All messages are displayed in the standard log.

Result	The message module functions <code>msg_xxx_set</code> and <code>msg_xxx_printf</code> always use the specified default dialog type. The dialog type of the functions <code>msg_set</code> and <code>msg_printf</code> is set to the default type when they are calling with the <code>msg_dialog</code> argument <code>MSG_DLG_DEFAULT</code> .
---------------	---

Parameters	msg_class Specifies the type of the message. The following symbols are predefined:
-------------------	---

Predefined Symbol	Meaning
<code>MSG_MC_ERROR</code>	Error message
<code>MSG_MC_INFO</code>	Information message

Predefined Symbol	Meaning
MSG_MC_WARNING	Warning message
MSG_MC_LOG	Message appears only in the log file

msg_dialog Specifies the type of the dialog. The following types are predefined:

Predefined Symbol	Meaning
MSG_DLG_NONE	No dialog, silent mode
MSG_DLG_OKCANCEL	OK/Cancel dialog

Return value None

Related topics Basics

[Basic Principles of Message Handling](#)..... 131

msg_mode_set

Syntax `void msg_mode_set(UINT32 mode)`

Include file `dsmsg.h`

Purpose To set the mode of the message buffer.

Description This function specifies the behavior of the message buffer if the number of messages exceeds the maximum buffer length. On start-up, the overwrite mode is active.

Parameters **mode** Specifies the mode of the message buffer. The following symbols are predefined:

Predefined Symbol	Meaning
MSG_BLOCKING	The message buffer will be filled to the maximum number of entries. Any further messages will be lost.
MSG_OVERWRITE	The message buffer will be filled cyclically. The oldest message will be overwritten when the buffer is full.

Return value	None
---------------------	------

Related topics	Basics
-----------------------	--------

[Basic Principles of Message Handling](#)..... 131

msg_reset

Syntax	<code>void msg_reset()</code>
---------------	-------------------------------

Include file	<code>dsmsg.h</code>
---------------------	----------------------

Purpose	To reset the message buffer and clear the values of the last error (see msg_error_clear).
----------------	--

Description	The next message will be the first entry in the message buffer. Nevertheless, the message number will be incremented.
--------------------	---

Return value	None
---------------------	------

Related topics	Basics
-----------------------	--------

[Basic Principles of Message Handling](#)..... 131

References

[msg_error_clear](#)..... 149

msg_last_error_number

Syntax	<code>msg_no_type msg_last_error_number()</code>
---------------	--

Include file dsmsg.h

Purpose To read the number of the last generated error message.

Description Independently of the order of the messages in the message buffer, this function returns the number of the last error message. On start-up, the value is set to 0.

Note

Warning and information messages do not change this number.

Return value This function returns the number of the last generated error message.

Related topics Basics

Basic Principles of Message Handling..... 131

References

msg_error_clear..... 149
msg_last_error_submodule..... 147

msg_last_error_submodule

Syntax msg_submodule_type msg_last_error_submodule()

Include file dsmsg.h

Purpose To read the submodule of the last generated error message.

Description	On start-up, the value is set to MSG_SM_NONE (see table below).
--------------------	--

Note

Warning and information messages do not change this value.

Return value	This function returns the submodule of the last generated error message. The following symbols are defined:
---------------------	---

Predefined Symbol	Message refers to ...
MSG_SM_NONE	No specific module (default)
MSG_SM_USER	User messages
MSG_SM_CAN1401	RTLib: CAN (DS1401)
MSG_SM_CAN2202	RTLib: CAN (DS2202)
MSG_SM_CAN2210	RTLib: CAN (DS2210)
MSG_SM_CAN2211	RTLib: CAN (DS2211)
MSG_SM_CAN4302	RTLib: CAN (DS4302)
MSG_SM_DIO1401	RTLib: Digital I/O (DS1401)
MSG_SM_DS1104SLVLIB	RTLib: Slave DSP (DS1104)
MSG_SM_DS4501	RTLib: DS4501 functions
MSG_SM_DS4502	RTLib: DS4502 functions
MSG_SM_DSBYPASS	RTI: Bypass Blockset
MSG_SM_DSCAN	RTLib: CAN support
MSG_SM_DSETH	RTI: RTI Ethernet Blockset
MSG_SM_DSFR	RTLib: FlexRay support
MSG_SM_DSJ1939	J1939 Support in RTI CAN MultiMessage Blockset
MSG_SM_DSSER	RTLib: Serial interface
MSG_SM_ECU_POD	ECU PODs (DS5xx)
MSG_SM_ECU1401	RTLib: ECU interface (DS1401)
MSG_SM_HOSTSERV	Host services
MSG_SM_LIN	RTLib: LIN support
MSG_SM_REALMOTION	RealMotion / MotionDesk
MSG_SM_RTII	Real-Time Interface
MSG_SM_RTICAN	RTI: CAN Blockset
MSG_SM_RTICAN1401	RTI: CAN Blockset (DS1401)
MSG_SM_RTICAN2202	RTI: CAN Blockset (DS2202)
MSG_SM_RTICAN2210	RTI: CAN Blockset (DS2210)
MSG_SM_RTICAN2211	RTI: CAN Blockset (DS2211)
MSG_SM_RTICAN4302	RTI: CAN Blockset (DS4302)

Predefined Symbol	Message refers to ...
MSG_SM_RTICANMM	RTI: CAN MultiMessage Blockset
MSG_SM_RTIFLEXRAY	RTI: FlexRay Blockset
MSG_SM_RTIFLEXRAYCONFIG	RTI: FlexRay Configuration Blockset
MSG_SM_RTILINMM	RTI: LIN MultiMessage Blockset
MSG_SM_RTIMP	RTI-MP (Real-Time Interface for multiprocessor systems)
MSG_SM_RTKERNEL	Real-Time Kernel
MSG_SM_RTLIB	Real-Time Board Library
MSG_SM_RTOSAL	RTOS Abstractionlayer
MSG_SM RTPYTHON	RTPython interpreter
MSG_SM_SIMENG	RTI: Simulation engine

Related topics**Basics**

Basic Principles of Message Handling 131

References

msg_error_clear 149

msg_last_error_number 146

msg_error_clear

Syntax`void msg_error_clear()`**Include file**

dsmsg.h

Purpose

To set the number of the last generated error to 0 and the submodule of the last generated error message to **MSG_SM_NONE** (refer to [Data Types and Symbols for Message Handling](#) on page 132).

Return value

None

Related topics**Basics**

Basic Principles of Message Handling	131
--	-----

References

msg_last_error_number	146
msg_last_error_submodule	147
msg_reset	146

msg_error_hook_set

Syntax

```
void msg_error_hook_set(msg_hookfcn_type hook)
```

Include file

dsmsg.h

Purpose

To install a hook function.

Description

The hook function is activated when an error message is generated (see [msg_error_set](#) and [msg_error_printf](#)) and before the message is displayed.

Use the hook function to:

- React to an error (for example, to implement an error correction function)
- Suppress the error message

The hook function is activated for all errors. To react only for certain submodules or message numbers, you have to manage restrictions within your handcoded function (see example below).

Parameters

hook Specifies the pointer to the hook function.

Return value

This function returns one of the following values:

Value	Meaning
1	The error message is displayed.
0	The error message is not displayed.

Example

This example shows how to use a hook function:

```
#include <Brtenv.h>
int error_hook_function(msg_submodule_type sm, msg_no_type no)
{
    if ((sm == MSG_SM_RTI) && (no == 1))
    {
        /* suppress error message */
        return(0);
    } else
    {
        /* display error message */
        return(1);
    }
}
void main()
{
    /* Initialization of the board */
    init();
    /* Announce the hook function to the message module */
    msg_error_hook_set(error_hook_function);
    /* Write an error message to the message buffer */
    msg_error_set(MSG_SM_USER, 1, "user error message");
    /* This error message will be suppressed by the
       hook function */
    msg_error_set(MSG_SM_RTI, 1, "RTI error message");
}
```

Related topics**Basics**

Basic Principles of Message Handling	131
--	-----

msg_init

Syntax

<code>void msg_init(void)</code>

Include file

<code>dsmsg.h</code>

Purpose

To initialize the message handling.

Description

This function is called automatically during the board initialization. The mode is set to MSG_OVERWRITE, counter and indices are set to 0. The buffer and string
--

lengths are set according to the values of `MSG_BUFFER_LENGTH` and `MSG_STRING_LENGTH` defined in `SrtkMsg.h`.

Return value	None
---------------------	------

Related topics	Basics
-----------------------	---------------

[Basic Principles of Message Handling](#)..... 131

References

[msg_mode_set](#)..... 145

System Functions

Introduction	Systems functions allow you to access board devices, such as LEDs, buzzer and the sensor supply.
---------------------	--

Where to go from here	Information in this section
	Buzzer Control153 With the buzzer control functions, you can implement acoustic signals in your application.
	LED Control162 With the LED control functions, you can set the color of the four customizable LEDs.
	Sensor Supply171 With the sensor supply functions, you can control the two voltage outputs.

Buzzer Control

Introduction	With the buzzer control functions, you can implement acoustic signals in your application.
---------------------	--

Where to go from here	Information in this section
	Buzzer_apply154 To apply the initialization settings to the buzzer.
	Buzzer_create155 To create the driver object for accessing a buzzer.
	Buzzer_setDuration156 To set the duration of one beep.
	Buzzer_setFrequency157 To set the frequency of a beep.
	Buzzer_setNumberOfBeeps158 To set the number of beeps.
	Buzzer_setPause160 To set the duration between two beeps of the buzzer.

Buzzer_start..... 161

To activate the buzzer channel.

Buzzer_write..... 161

To update settings of the buzzer during run time of the real-time application.

Information in other sections

Implementing I/O Functions..... 198

Buzzer_apply

Syntax

```
UInt32 Buzzer_apply(BuzzerSDrvObject *pBuzzer)
```

Include file

IoDrvBuzzer.h

Purpose

To apply the initialization settings to the buzzer channel.

Description

Before you can start the specified buzzer, you have to transfer the settings of the buzzer driver object to its related output channel by using **Buzzer_apply**. You have to do this also for the default values and for values that you specified by using the **Buzzer_set<Parameter>** functions. The **Buzzer_apply** function is intended to be called at the end of the initialization phase of the real-time application.

The configured acoustic signal is not generated until you have called **Buzzer_start**.

To update buzzer settings during run time, you have to use **Buzzer_write**.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

pBuzzer Lets you specify the buzzer to be used. The buzzer driver object must already have been created by using **Buzzer_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example

This example shows how to apply the settings of the buzzer:

```
UInt32 IError;
IError = Buzzer_apply(MyBuzzer);
```

Related topics**References**

Buzzer_create.....	155
Buzzer_setDuration.....	156
Buzzer_setFrequency.....	157
Buzzer_setNumberOfBeeps.....	158
Buzzer_setPause.....	160
Buzzer_start.....	161
Buzzer_write.....	161

Buzzer_create

Syntax

```
UInt32 Buzzer_create(
    BuzzerSDrvObject **ppBuzzer,
    UInt32 Channel)
```

Include file

IoDrvBuzzer.h

Purpose

To create the driver object for accessing a buzzer.

Description

This function performs all the steps necessary to create a buzzer driver object. The buzzer is initialized with default values so that the driver is ready for use.

The initial values of the buzzer are:

- Frequency: 892 Hz
- Beep duration: 1 s
- Pause: 10 ms
- Number of beeps: 1

The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

ppBuzzer Lets you specify the address of a variable which holds the address of the created driver object.

Channel Lets you select BUZZER_CHANNEL_1 as the output channel to be controlled by this driver. Currently, only one channel is available for the buzzer.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example This example shows how to create a buzzer driver object:

```
UInt32 IError;
BuzzerSDrvObject* MyBuzzer;
IError = Buzzer_create(&MyBuzzer, BUZZER_CHANNEL_1);
```

Related topics

References

Buzzer_apply.....	154
-------------------	-----

Buzzer_setDuration

Syntax

```
UInt32 Buzzer_setDuration(
    BuzzerSDrvObject *pBuzzer,
    Float64 Duration)
```

Include file

IoDrvBuzzer.h

Purpose

To set the duration of one beep.

Description

This function sets the duration of one beep of the acoustic signal to be output by the buzzer.

The value does not take effect until you have called **Buzzer_apply** during the initialization phase or **Buzzer_write** during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Parameters

pBuzzer Lets you specify the buzzer to be used. The buzzer driver object must already have been created by using **Buzzer_create**.

Duration Lets you specify the duration of one beep in s in the range 0.01 ... 2.54 s. You can specify the value in steps of 10 ms. If you specify a value greater than 2.54 s, the beep is generated permanently.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example

This example shows how to set the duration of one beep to 1.5 seconds:

```
UInt32 IError;
IError = Buzzer_setDuration(MyBuzzer, 1.5);
```

Related topics**References**

Buzzer_apply.....	154
Buzzer_setFrequency.....	157
Buzzer_setNumberOfBeeps.....	158
Buzzer_setPause.....	160
Buzzer_write.....	161

Buzzer_setFrequency

Syntax

```
UInt32 Buzzer_setFrequency(
    BuzzerSDrvObject *pBuzzer,
    Float64 Frequency)
```

Include file

IoDrvBuzzer.h

Purpose	To set the frequency of a beep.										
Description	<p>This function sets the frequency of the acoustic signal to be output by the buzzer.</p> <p>The value does not take effect until you have called Buzzer_apply during the initialization phase or Buzzer_write during run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>										
Parameters	<p>pBuzzer Lets you specify the buzzer to be used. The buzzer driver object must already have been created by using Buzzer_create.</p> <p>Frequency Lets you specify the frequency of the buzzer in Hz in the range 98 Hz ... 25 kHz. You can specify the value in steps of 40 µs.</p>										
Return value	The function returns an error code.										
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.				
Error Code	Meaning										
IOLIB_NO_ERROR	The function was successfully completed.										
!= IOLIB_NO_ERROR	The function was not completed.										
Example	This example shows how to set the frequency of a beep to 440 Hz:										
	<pre>UInt32 IError; IError = Buzzer_setFrequency(MyBuzzer, 440);</pre>										
Related topics	References										
	<table> <tbody> <tr> <td>Buzzer_apply.....</td><td>154</td></tr> <tr> <td>Buzzer_setDuration.....</td><td>156</td></tr> <tr> <td>Buzzer_setNumberOfBeeps.....</td><td>158</td></tr> <tr> <td>Buzzer_setPause.....</td><td>160</td></tr> <tr> <td>Buzzer_write.....</td><td>161</td></tr> </tbody> </table>	Buzzer_apply.....	154	Buzzer_setDuration.....	156	Buzzer_setNumberOfBeeps.....	158	Buzzer_setPause.....	160	Buzzer_write.....	161
Buzzer_apply.....	154										
Buzzer_setDuration.....	156										
Buzzer_setNumberOfBeeps.....	158										
Buzzer_setPause.....	160										
Buzzer_write.....	161										

Buzzer_setNumberOfBeeps

Syntax	<pre>UInt32 Buzzer_setNumberOfBeeps(BuzzerSDrvObject *pBuzzer, UInt32 NumberOfBeeps)</pre>
---------------	---

Include file	IoDrvBuzzer.h										
Purpose	To set the number of beeps.										
Description	<p>This function sets the number of beeps to be output by the buzzer. The value does not take effect until you have called Buzzer_apply during the initialization phase or Buzzer_write during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.</p>										
Parameters	<p>pBuzzer Lets you specify the buzzer to be used. The buzzer driver object must already have been created by using Buzzer_create.</p> <p>NumberOfBeeps Lets you specify the number of beeps in the range 0 ... 255. If you specify 0, no acoustic signal will be output, if you specify 255, the number of beeps is infinite. To stop an infinite beep, update this value to 0 or another value less than 255.</p>										
Return value	The function returns an error code.										
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.				
Error Code	Meaning										
IOLIB_NO_ERROR	The function was successfully completed.										
!= IOLIB_NO_ERROR	The function was not completed.										
Example	<p>This example shows how to set the number of beeps to 10:</p> <pre>UInt32 IError; IError = Buzzer_setNumberOfBeeps(MyBuzzer, 10);</pre>										
Related topics	References <table border="1"> <tbody> <tr> <td>Buzzer_apply.....</td><td>154</td></tr> <tr> <td>Buzzer_setDuration.....</td><td>156</td></tr> <tr> <td>Buzzer_setFrequency.....</td><td>157</td></tr> <tr> <td>Buzzer_setPause.....</td><td>160</td></tr> <tr> <td>Buzzer_write.....</td><td>161</td></tr> </tbody> </table>	Buzzer_apply.....	154	Buzzer_setDuration.....	156	Buzzer_setFrequency.....	157	Buzzer_setPause.....	160	Buzzer_write.....	161
Buzzer_apply.....	154										
Buzzer_setDuration.....	156										
Buzzer_setFrequency.....	157										
Buzzer_setPause.....	160										
Buzzer_write.....	161										

Buzzer_setPause

Syntax

```
UInt32 Buzzer_setPause(
    BuzzerSDrvObject *pBuzzer,
    Float64 Pause)
```

Include file

`IoDrvBuzzer.h`

Purpose

To set the duration between two beeps of the buzzer.

Description

This function sets the duration of a pause between two beeps of the buzzer.

The value does not take effect until you have called `Buzzer_apply` during the initialization phase or `Buzzer_write` during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Parameters

pBuzzer Lets you specify the buzzer to be used. The buzzer driver object must already have been created by using `Buzzer_create`.

Pause Lets you specify the duration of the pause between two beeps of the buzzer in s in the range 10 ms ... 2.55 s. You can specify the value in steps of 10 ms.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Example

This example shows how to set the duration between two beeps to 2 seconds:

```
UInt32 IError;
IError = Buzzer_setPause(MyBuzzer, 2.0);
```

Related topics

References

<code>Buzzer_apply</code>	154
<code>Buzzer_setDuration</code>	156
<code>Buzzer_setFrequency</code>	157

Buzzer_setNumberOfBeeps.....	158
Buzzer_write.....	161

Buzzer_start

Syntax

```
UInt32 Buzzer_start(BuzzerSDrvObject *pBuzzer)
```

Include file

IoDrvBuzzer.h

Purpose

To activate the buzzer channel.

Description

This function starts the given buzzer channel. It is required to output the specified acoustic signal.

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

Parameters

pBuzzer Lets you specify the buzzer to be used. The buzzer driver object must already have been created by using [Buzzer_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Buzzer_create.....	155
--------------------	-----

Buzzer_write

Syntax

```
UInt32 Buzzer_write(BuzzerSDrvObject *pBuzzer)
```

Include file	IoDrvBuzzer.h						
Purpose	To update settings of the buzzer during run time of the real-time application.						
Description	<p>The functions for configuring the acoustic signal of the buzzer can also be used during run time of the real-time application.</p> <p>To write the settings to the output channel during run time, you have to use Buzzer_write. The acoustic signal is not output until you have called Buzzer_start once.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
Parameters	<p>pBuzzer Lets you specify the buzzer to be used. The buzzer driver object must already have been created by using Buzzer_create.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"><thead><tr><th>Error Code</th><th>Meaning</th></tr></thead><tbody><tr><td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr><tr><td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr></tbody></table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	References <table><tr><td>Buzzer_create.....</td><td>155</td></tr><tr><td>Buzzer_start.....</td><td>161</td></tr></table>	Buzzer_create	155	Buzzer_start	161		
Buzzer_create	155						
Buzzer_start	161						

LED Control

Introduction	With the LED control functions, you can set the color of the four customizable LEDs.
---------------------	--

Where to go from here**Information in this section**

Led_apply	163
To apply the color settings to an LED.	
Led_create	164
To create the driver object for accessing an LED.	
Led_setBlueColor	165
To set the blue content of the LED's color.	
Led_setGreenColor	167
To set the green content of the LED's color.	
Led_setRedColor	168
To set the red content of the LED's color.	
Led_start	169
To activate an LED channel.	
Led_write	170
To update the color settings of an LED during run time.	

Information in other sections

Implementing I/O Functions	198
--	-----

Led_apply

Syntax

```
UInt32 Led_apply(LedDrvObject *pLed)
```

Include file

IoDrvLed.h

Purpose

To apply the color settings to an LED.

Description

Before you can start the specified LED, you must transfer the settings of the LED driver object to its related output channel by using `Led_apply`. You have to do this also for the default values and for values that you specified by using the `Led_set<Parameter>` functions. The `Led_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The configured LED color is not generated until you have called `Led_start`.

To update the LED color settings during run time, you must use **Led_write**.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

pLed Lets you specify the LED to be used. The LED driver object must already have been created by using **Led_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Led_create	164
Led_start	169
Led_write	170

Led_create

Syntax

```
UInt32 Led_create(
    LedSDrvObject **ppLed,
    UInt32 Channel)
```

Include file

IoDrvLed.h

Purpose

To create the driver object for accessing an LED.

Description

This function performs all the steps necessary to create an LED driver object for controlling one of the four LEDs. The LED is initialized with default values for the RGB color value so that the driver is ready for use.

The initial values of the LED are:

- Red color: 0 (off)
- Green color: 0 (off)
- Blue color: 0 (off)

The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

ppLed Lets you specify the address of a variable which holds the address of the created driver object.

Channel Lets you select the LED to be controlled by specifying the related output channel of the LED driver.

Symbol	Meaning
LED_CHANNEL_1	Specifies channel 1 that controls LED 1.
...	...
LED_CHANNEL_4	Specifies channel 4 that controls LED 4.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example

This example shows how to create an LED driver object to control LED 2:

```
UInt32 IError;
LedSDrvObject* MyLed;
IError = Led_create(&MyLed, LED_CHANNEL_2);
```

Led_setBlueColor

Syntax

```
UInt32 Led_setBlueColor(
    LedSDrvObject *pLed,
    UInt32 BlueColor)
```

Include file

IoDrvLed.h

Purpose

To set the blue content of the LED's color.

Description

This function sets the blue content of the LED's RGB value. The combination of the values specified by using `Led_setRedColor`, `Led_setGreenColor` and `Led_setBlueColor` results in the color the LED will light.

The value does not take effect until you have called `Led_apply` during the initialization phase or `Led_write` during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

pLed Lets you specify the LED to be used. The LED driver object must already have been created by using `Led_create`.

BlueColor Lets you specify the blue content of the RGB color value of the LED in the range 0 ... 255. The maximum intensity is specified by 255.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example

This example shows how to set the blue content of the LED color to 150:

```
UInt32 IError;
IError = Led_setBlueColor(MyLed, 150);
```

If you want the LED to light yellow with the RGB value of (255,255,0), the other colors must be set to 0 and 255:

```
UInt32 IError;
IError = Led_setRedColor(MyLed, 255);
IError = Led_setGreenColor(MyLed, 255);
IError = Led_setBlueColor(MyLed, 0);
```

Related topics

References

<code>Led_apply</code>	163
<code>Led_setGreenColor</code>	167
<code>Led_setRedColor</code>	168
<code>Led_write</code>	170

Led_setGreenColor

Syntax

```
UInt32 Led_setGreenColor(
    LedSDrvObject *pLed,
    UInt32 GreenColor)
```

Include file

IoDrvLed.h

Purpose

To set the green content of the LED's color.

Description

This function sets the green content of the LED's RGB value. The combination of the values specified by using `Led_setRedColor`, `Led_setGreenColor` and `Led_setBlueColor` results in the color the LED will light.

The value does not take effect until you have called `Led_apply` during the initialization phase or `Led_write` during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

pLed Lets you specify the LED to be used. The LED driver object must already have been created by using `Led_create`.

GreenColor Lets you specify the green content of the LED's RGB color value in the range 0 ... 255. The maximum intensity is specified by 255.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example

This example shows how to set the green content of the LED color to 150:

```
UInt32 IError;
IError = Led_setGreenColor(MyLed, 150);
```

If you want the LED to light green with the RGB value of (0,255,0), the other colors must be set to 0:

```
UInt32 IError;
IError = Led_setRedColor(MyLed, 0);
IError = Led_setGreenColor(MyLed, 255);
IError = Led_setBlueColor(MyLed, 0);
```

Related topics

References

Led_apply.....	163
Led_setBlueColor.....	165
Led_setRedColor.....	168
Led_write.....	170

[Led_setRedColor](#)

Syntax

```
UInt32 Led_setRedColor(
    LedDrvObject *pLed,
    UInt32 RedColor)
```

Include file

IoDrvLed.h

Purpose

To set the red content of the LED's color.

Description

This function sets the red content of the LED's RGB value. The combination of the values specified by using `Led_setRedColor`, `Led_setGreenColor` and `Led_setBlueColor` results in the color the LED will light.

The value does not take effect until you have called `Led_apply` during the initialization phase or `Led_write` during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

pLed Lets you specify the LED to be used. The LED driver object must already have been created by using `Led_create`.

RedColor Lets you specify the red content of the LED's RGB color value in the range 0 ... 255. The maximum intensity is specified by 255.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example

This example shows how to set the red content of the LED color to 150:

```
UInt32 IError;
IError = Led_setRedColor(MyLed, 150);
```

If you want the LED to light red with the RGB value of (255,0,0), the other colors must be set to 0:

```
UInt32 IError;
IError = Led_setRedColor(MyLed, 255);
IError = Led_setGreenColor(MyLed, 0);
IError = Led_setBlueColor(MyLed, 0);
```

Related topics**References**

Led_apply.....	163
Led_setBlueColor.....	165
Led_setGreenColor.....	167
Led_write.....	170

Led_start

Syntax

```
UInt32 Led_start(LedSDrvObject *pLed)
```

Include file

IoDrvLed.h

Purpose

To activate an LED channel.

Description

This function starts the specified LED channel. The LED does not light until you have called this function.

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

Parameters

pLed Lets you specify the LED to be used. The LED driver object must already have been created by using **Led_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Led_create.....	164
-----------------	-----

Led_write

Syntax

```
UInt32 Led_write(LedsDrvObject *pLed)
```

Include file

IoDrvLed.h

Purpose

To update the color settings of an LED during run time.

Description

The functions for configuring the color settings of an LED can also be used during run time of the real-time application. To write the settings to the output channel during run time, you must use **Led_write**.

The LED does not light until you have called **Led_start**.

If an error is detected, the very first instance of the error is returned to the caller.

Parameters

pLed Lets you specify the LED to be used. The LED driver object must already have been created by using **Led_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Sensor Supply

Introduction

With the sensor supply functions, you can control the two voltage outputs.

Note

One of the sensor supplies provides a constant output voltage. It is permanently available and you do not need to implement access functions for it. The following functions are only required to use the adjustable sensor supply. For further information, refer to [SensorSupply_create](#) on page 173.

Where to go from here

Information in this section

SensorSupply_apply	171
To apply the initialization settings to the sensor supply.	
SensorSupply_create	173
To create the driver object for accessing a sensor supply.	
SensorSupply_setSensorState	174
To set the sensor supply state.	
SensorSupply_setVoltage	175
To set the voltage value to be output by the sensor supply.	
SensorSupply_start	177
To activate the voltage out generation on the specified sensor supply.	
SensorSupply_write	178
To write the settings to the sensor supply.	

Information in other sections

Implementing I/O Functions	198
--	-----

SensorSupply_apply

Syntax

```
UInt32 SensorSupply_apply(SensorSupplySDrvObject *pSensor)
```

Include file

IoDrvSensorSupply.h

Purpose	To apply the initialization settings to the sensor supply.						
Description	<p>Before you can start the specified sensor supply, you must transfer the settings of the sensor supply driver object to its related output channel by using <code>SensorSupply_apply</code>. You must do this also for the default values and for values that you specified by using the <code>SensorSupply_set<Parameter></code> functions. The <code>SensorSupply_apply</code> function is intended to be called at the end of the initialization phase of the real-time application.</p> <p>The configured output voltage is not generated until you have called <code>SensorSupply_start</code>.</p> <p>To update the output voltage during run time, you must use <code>SensorSupply_write</code>.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Sensor Supply (MicroLabBox Features)</p>						
Parameters	<p><code>pSensor</code> Lets you specify the sensor supply to be used. The sensor supply driver object must already have been created by using <code>SensorSupply_create</code>.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"><thead><tr><th>Error Code</th><th>Meaning</th></tr></thead><tbody><tr><td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr><tr><td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr></tbody></table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						
Related topics	References						
	<p>SensorSupply_create..... 173</p>						

SensorSupply_create

Syntax

```
UInt32 SensorSupply_create(
    SensorSupplySDrvObject **ppSensor,
    UInt32 SensorNumber)
```

Include file

`IoDrvSensorSupply.h`

Purpose

To create the driver object for accessing a sensor supply.

Description

This function performs all the steps necessary to create a sensor supply driver object for accessing one of the two sensor supplies for voltage output. The sensor supply is initialized with default values for the output voltage so that the driver is ready for use.

The initial values of the sensor supply are:

- Output voltage: 2.0 V (for the adjustable sensor)
- Sensor supply state: disabled

The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Sensor Supply \(MicroLabBox Features\)](#)

Parameters

ppSensor Lets you specify the address of a variable that provides the address of the created sensor supply driver object.

SensorNumber Lets you specify the sensor to be controlled by the sensor supply driver.

Predefined Symbol	Meaning
SENSOR_SUPPLY_NO_1	<p>The sensor supply driver object controls sensor 1. This is the sensor supply with the constant voltage output.</p> <p>You have to note:</p> <ul style="list-style-type: none"> ▪ Calling <code>setVoltage</code> has not effect. ▪ The voltage out is generated directly after instantiating the driver. Calling <code>start</code> has only an

Predefined Symbol	Meaning
SENSOR_SUPPLY_NO_2	effect if you want to restart the outputs after setting their states to inactive. The sensor supply driver object controls sensor 2. This is the sensor supply with the adjustable voltage output.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example

This example shows how to create the I/O driver object for the adjustable sensor supply 2:

```
UInt32 IError;
SensorSupplySDrvObject* MySensorSupply;
IError = SensorSupply_create(&MySensorSupply, SENSOR_SUPPLY_NO_2);
```

Related topics**References**

SensorSupply_apply.....	171
-------------------------	-----

SensorSupply_setSensorState

Syntax

```
UInt32 SensorSupply_setSensorState(
    SensorSupplySDrvObject *pSensor,
    UInt32 State)
```

Include file

IoDrvSensorSupply.h

Purpose

To set the sensor supply state.

Description

This function sets the run-time state of the specified sensor supply. The initially disabled sensor supply state is enabled if you use **SensorSupply_start**. You

can disable the sensor supply, for example, to set it to a definite state when terminating the real-time application.

The value does not take effect until you have called **SensorSupply_apply** during the initialization phase or **SensorSupply_write** during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Sensor Supply \(MicroLabBox Features\)](#)

Parameters

pSensor Lets you specify the sensor supply to be used. The sensor supply driver object must already have been created by using **SensorSupply_create**.

State Lets you set the state of the specified sensor supply.

Predefined Symbol	Meaning
SENSOR_STATE_DISABLE	Disables the sensor supply functionality.
SENSOR_STATE_ENABLE	Enables the sensor supply functionality.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

SensorSupply_apply	171
SensorSupply_write	178

SensorSupply_setVoltage

Syntax

```
UInt32 SensorSupply_setVoltage(
    SensorSupplySDrvObject *pSensor,
    Float64 VoltageOut)
```

Include file	IoDrvSensorSupply.h						
Purpose	To set the voltage value to be output by the sensor supply.						
Description	This function sets the output voltage of the adjustable sensor supply. <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p>Note</p><p>The sensor supply with the fixed output voltage does not react to this setting. The sensor supply is selected by the <code>SensorNumber</code> parameter of the <code>SensorSupply_create</code> function.</p></div>						
<p>The value does not take effect until you have called <code>SensorSupply_apply</code> during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>							
I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) . For more information on the I/O mapping, refer to Sensor Supply (MicroLabBox Features)						
Parameters	<p>pSensor Lets you specify the sensor supply to be used. The sensor supply driver object must already have been created by using <code>SensorSupply_create</code>.</p> <p>VoltageOut Lets you specify the output voltage in V to be generated on the adjustable sensor supply in the range +2.0 ... +20.0 V. The resolution of the sensor supply is 5 mV. If the specified output voltage is lower than the minimum value, the value is set to 2.0 V. If the specified output voltage is greater than the maximum value, the value is set to 20.0 V.</p>						
Return value	The function returns an error code. <table border="1" style="width: 100%; border-collapse: collapse;"><thead><tr><th>Error Code</th><th>Meaning</th></tr></thead><tbody><tr><td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr><tr><td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr></tbody></table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Example

This example shows how to set the output voltage of the adjustable sensor supply to 12.0 V:

```
UInt32 IError;
IError = SensorSupply_setVoltage(MySensorSupply, 12.0);
```

Related topics**References**

SensorSupply_apply.....	171
SensorSupply_create.....	173

SensorSupply_start

Syntax

```
UInt32 SensorSupply_start(SensorSupplySDrvObject *pSensor)
```

Include file

`IoDrvSensorSupply.h`

Purpose

To activate the voltage out generation on the specified sensor supply.

Description

This function starts the specified sensor supply driver and enables the sensor outputs used for voltage generation.

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Sensor Supply \(MicroLabBox Features\)](#)

Parameters

pSensor Lets you specify the sensor supply to be used. The sensor supply driver object must already have been created by using [SensorSupply_create](#).

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References				
	<table> <tr> <td>SensorSupply_create.....</td> <td>173</td> </tr> <tr> <td>SensorSupply_setSensorState.....</td> <td>174</td> </tr> </table>	SensorSupply_create.....	173	SensorSupply_setSensorState.....	174
SensorSupply_create.....	173				
SensorSupply_setSensorState.....	174				

SensorSupply_write

Syntax	<code>UInt32 SensorSupply_write(SensorSupplySDrvObject *pSensor)</code>
Include file	<code>IoDrvSensorSupply.h</code>
Purpose	To write the settings to the sensor supply.
Description	<p>This function updates the run-time settings of the specified sensor supply. The settings are not output until you have called SensorSupply_start. The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Sensor Supply (MicroLabBox Features)</p>
Parameters	pSensor Lets you specify the sensor supply to be used. The sensor supply driver object must already have been created by using SensorSupply_create .

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

SensorSupply_create.....	173
SensorSupply_setSensorState.....	174
SensorSupply_start.....	177

Special Processor Functions

Purpose	To ensure proper operation of the PowerPC.
Where to go from here	Information in this section

RTLIB_FORCE_IN_ORDER	180
To force the processor to do the last I/O access in order.	
RTLIB_SYNC	181
To force the processor to perform all pending memory accesses.	

RTLIB_FORCE_IN_ORDER

Syntax	<code>void RTLIB_FORCE_IN_ORDER(void)</code>
Include file	<code>SrtkStd.h</code>
Purpose	To force the processor to execute the I/O accesses in order.
Description	This macro ensures that the PowerPC executes I/O accesses in the right order. For example, when two I/O accesses are performed sequentially, the PowerPC can change their order. If the <code>RTLIB_FORCE_IN_ORDER</code> macro is executed between the two accesses, they are executed in the specified order.
Return value	None
Related topics	References

RTLIB_SYNC	181
----------------------------------	-----

RTLIB_SYNC

Syntax

```
void RTLIB_SYNC(void)
```

Include file

SrtkStd.h

Purpose

To force the PowerPC to perform all pending memory accesses.

Description

This macro ensures that the PowerPC performs all memory accesses that were issued before the macro was called.

Return value

None

Related topics**References**

RTLIB_FORCE_IN_ORDER.....	180
---------------------------	-----

Conversion Functions

Introduction

Use these macros to convert floating-point values to other formats. Conversion is necessary because the PowerPC and the TI slave processors on the I/O boards use different floating-point formats. The PowerPC uses the IEEE floating-point format, and the TI slave processor uses the TI floating-point format. TI floating-point values are stored as UInt32 because they are usually transferred to the external hardware through 32-bit I/O registers.

Where to go from here

Information in this section

RTLIB_CONV_FLOAT32_FROM_IEEE32.....	182
RTLIB_CONV_FLOAT32_FROM_TI32.....	183
RTLIB_CONV_FLOAT32_TO_IEEE32.....	183
RTLIB_CONV_FLOAT_TO_SATURATED_INT32.....	184
RTLIB_CONV_FLOAT32_TO_TI32.....	184

RTLIB_CONV_FLOAT32_FROM_IEEE32

Syntax

```
Float32 RTLIB_CONV_FLOAT32_FROM_IEEE32(UInt32 ieee_32)
```

Include file

SrtkStd.h

Purpose

To convert a value in IEEE floating-point format to native floating-point format.

Parameters

ieee_32 Specifies the value in IEEE floating-point format.

Return value

This function returns the value in native floating-point format.

Related topics

References

RTLIB_CONV_FLOAT32_TO_IEEE32.....	183
-----------------------------------	-----

RTLIB_CONV_FLOAT32_FROM_TI32

Syntax

```
Float32 RTLIB_CONV_FLOAT32_FROM_TI32(UInt32 ti_32)
```

Include file

SrtkStd.h

Purpose

To convert a value in TI floating-point format to IEEE floating-point format.

Parameters

ti_32 Specifies the value in TI floating-point format.

Return value

This function returns the value in IEEE floating-point format.

Related topics**References**

[RTLIB_CONV_FLOAT32_TO_TI32](#)..... 184

RTLIB_CONV_FLOAT32_TO_IEEE32

Syntax

```
UInt32 RTLIB_CONV_FLOAT32_TO_IEEE32(Float32 val_32)
```

Include file

SrtkStd.h

Purpose

To convert a value in native floating-point format to IEEE floating-point format.

Parameters

val_32 Specifies the value in float32 format.

Return value

This function returns the value in IEEE floating-point format.

Related topics**References**

[RTLIB_CONV_FLOAT32_FROM_IEEE32](#)..... 182

RTLIB_CONV_FLOAT_TO_SATURATED_INT32

Syntax

```
Int32 RTLIB_CONV_FLOAT_TO_SATURATED_INT32(double fp_value)
```

Include file

SrtkStd.h

Purpose

To convert a value in floating-point format to signed integer format.

Parameters

fp_value Specifies the value in floating-point format (float or double).

Return value

This function returns the value in signed integer format, possibly saturated.

Related topics**References**

RTLIB_CONV_FLOAT32_FROM_IEEE32.....	182
RTLIB_CONV_FLOAT32_TO_IEEE32.....	183

RTLIB_CONV_FLOAT32_TO_TI32

Syntax

```
UInt32 RTLIB_CONV_FLOAT32_TO_TI32(Float32 ieee_32)
```

Include file

SrtkStd.h

Purpose

To convert a value in IEEE floating-point format to TI floating-point format.

Parameters

ieee_32 Specifies the value in IEEE floating-point format.

Return value	This function returns the value in TI floating-point format.
Related topics	References

[RTLIB_CONV_FLOAT32_FROM_TIB2.....](#) 183

Standard Macros

Introduction	The include file <code>SrtkStd.h</code> defines several macros that can be used to program board-independent applications. For further information about the functionality of a macro, see either this topic or the description of the corresponding function.
Initialization	There is a macro to call the board-specific initialization routines. <ul style="list-style-type: none">▪ RTLIB_INIT on page 188
Application background	There is a macro that can be used to start all board-specific background functions. There are also standard functions for calling hook functions that are to run in the background of the application. <ul style="list-style-type: none">▪ RTLIB_BACKGROUND_SERVICE on page 29▪ rtlib_background_hook on page 29
End of application	There is a macro that can be used to terminate the application, for example, because of critical exceptions. Do not use it for a normal stop of an application. <ul style="list-style-type: none">▪ RTLIB_TERMINATE on page 189
Registering hook functions	There are macros that you can use to register functions that are to be called in the background service, the termination phase, or when you unload an application. <ul style="list-style-type: none">▪ RTLIB_REGISTER_BACKGROUND_HANDLER on page 190▪ RTLIB_REGISTER_TERMINATE_HANDLER on page 191▪ RTLIB_REGISTER_UNLOAD_HANDLER on page 191
Reading the board's serial number	There is a macro that you can use to get the serial number of your board. <ul style="list-style-type: none">▪ RTLIB_GET_SERIAL_NUMBER() on page 189
Interrupt handling	There are macros that can be used to enable or disable the interrupts globally. <ul style="list-style-type: none">▪ RTLIB_INT_ENABLE on page 106▪ RTLIB_INT_DISABLE on page 106
Sampling rate timer	There are macros to handle the default sampling rate timer. This is usually Timer A. <ul style="list-style-type: none">▪ RTLIB_SRT_START on page 91▪ RTLIB_SRT_PERIOD on page 66▪ RTLIB_SRT_ISR_BEGIN on page 90

- [RTLlib_SRT_ISR_END](#) on page 91
- [RTLlib_SRT_ENABLE](#) on page 109
- [RTLlib_SRT_DISABLE](#) on page 108

Time interval measurement There are macros to be used for time interval measurement.

Floating-point conversion There are macros to be used when converting floating-point values transferred from or to a TI slave processor of an I/O Board:

- [RTLlib_CONV_FLOAT32_TO_TI32](#) on page 184
- [RTLlib_CONV_FLOAT32_FROM_TI32](#) on page 183
- [RTLlib_CONV_FLOAT32_TO_IEEE32](#) on page 183
- [RTLlib_CONV_FLOAT32_FROM_IEEE32](#) on page 182
- [RTLlib_CONV_FLOAT_TO_SATURATED_INT32](#) on page 184

Memory allocation There are macros to handle memory allocation that is protected against interrupt activities.

- [RTLlib_MALLOC_PROT](#) on page 192
- [RTLlib_CALLOC_PROT](#) on page 193
- [RTLlib_REALLOC_PROT](#) on page 194
- [RTLlib_FREE_PROT](#) on page 194

Processor functions There are macros to handle the following Assembler commands.

- [RTLlib_FORCE_IN_ORDER](#) on page 180
- [RTLlib_SYNC](#) on page 181

init()

Purpose To initialize the required hardware and software modules for a specific hardware system.

Note

It is recommended to use `RTLlib_INIT` for new applications to avoid naming conflicts with `init` functions in other software modules.

Syntax

```
void init(void)
```

Include file	<code>Brtenv.h</code>
--------------	-----------------------

Description	This macro calls the internal initialization functions of the hardware system.
-------------	--

Note

- The initialization function `init()` must be executed at the beginning of each application. It can only be invoked once. Further calls to `init()` are ignored.
- When you use RTI, this function is called automatically in the simulation engine. Hence, you do not need to call `init()` in S-functions. If you need to initialize single components that are not initialized by `init()`, use the specific initialization functions that are described at the beginning of the function references.

RTLIB_INIT

Purpose	To initialize the required hardware and software modules for a specific hardware system.
---------	--

Syntax	<code>void RTLIB_INIT(void)</code>
--------	------------------------------------

Include file	<code>SrtkStd.h</code>
--------------	------------------------

Description	This macro calls the internal initialization functions of the specified hardware system.
-------------	--

Note

- The initialization function `RTLIB_INIT()` must be executed at the beginning of each application. It can only be invoked once. Further calls to `RTLIB_INIT()` are ignored.
- When you use RTI, this function is called automatically in the simulation engine. Hence, you do not need to call `RTLIB_INIT()` in S-functions. If you need to initialize single components that are not initialized by `RTLIB_INIT()`, use the specific initialization functions that are described at the beginning of the function references.

RTLIB_TERMINATE

Purpose To terminate the application.

Syntax `void RTLIB_TERMINATE(void)`

Include file SrtkStd.h

Description Usually, a real-time application is stopped by the host PC. You can handle this behavior by implementing termination code, but only if a critical error occurs during run time of the application.

A stopped application remains in the memory (RAM or flash) of the hardware and can be started again immediately. A terminated application also remains in the memory of the hardware, you must first reload it to restart it.

The RTLIB_TERMINATE macro behaves in the same way as the `exit` function, or a `main` function that finishes with a `return` statement. The exit code or the return code are ignored.

Note

To register a function that is to be executed in the termination phase, you must use `RTLIB_REGISTER_TERMINATE_HANDLER`. Do not use `atexit`.

Functions that are to be executed when you unload an application can be registered by using `RTLIB_REGISTER_UNLOAD_HANDLER`.

Related topics

References

RTLIB_REGISTER_TERMINATE_HANDLER.....	191
RTLIB_REGISTER_UNLOAD_HANDLER.....	191

RTLIB_GET_SERIAL_NUMBER()

Purpose To get the serial number of the processor board.

Syntax `RTLIB_GET_SERIAL_NUMBER()`

Include file SrtkStd.h

Description This macro returns the serial number as UInt32 data type.

RTLIB_REGISTER_BACKGROUND_HANDLER

Purpose To register a function that is called by the background service.

Syntax `Int32 RTLIB_REGISTER_BACKGROUND_HANDLER(SrtkTApp_Handler pHandler)`

Include file SrtkStd.h

Description You can use multiple calls of this macro to register more than one function. The functions are then executed in their registration order.

The registered background function should not contain endless loops or functions that might block the application.

Parameters **pHandler** Specifies the pointer to the handler of the function to be registered. The handler function must be of `SrtkTApp_Handler` type, i.e., `void MyBackgroundHandler(void)`.

Return value This function returns an error code.

Symbol	Meaning
SRTK_ERROR	The registration failed.
SRTK_NO_ERROR	The registration was performed without error.

Related topics**References**

RTLIB_REGISTER_TERMINATE_HANDLER.....	191
RTLIB_REGISTER_UNLOAD_HANDLER.....	191

RTLIB_REGISTER_TERMINATE_HANDLER

Purpose To register a function that is called in the termination phase.

Syntax

<code>Int32 RTLIB_REGISTER_TERMINATE_HANDLER(SrtkTApp_Handler pHandler)</code>
--

Include file `SrtkStd.h`

Description You can use multiple calls of this macro to register more than one function. The functions are then executed in the reverse order of their registration.

The registered termination function should not contain endless loops or functions that might block the application.

Parameters `pHandler` Specifies the pointer to the handler of the function to be registered. The handler function must be of `SrtkTApp_Handler` type, i.e., `void MyBackgroundHandler(void)`.

Return value This function returns an error code.

Symbol	Meaning
<code>SRTK_ERROR</code>	The registration failed.
<code>SRTK_NO_ERROR</code>	The registration was performed without error.

Related topics

References

<code>RTLIB_REGISTER_BACKGROUND_HANDLER</code>	190
<code>RTLIB_REGISTER_UNLOAD_HANDLER</code>	191

RTLIB_REGISTER_UNLOAD_HANDLER

Purpose To register a function that is called when you unload an application.

Syntax

<code>Int32 RTLIB_REGISTER_UNLOAD_HANDLER(SrtkTApp_Handler pHandler)</code>

Include file	SrtkStd.h						
Description	<p>You can use multiple calls of this macro to register more than one function. The functions are then executed in the reverse order of their registration.</p> <p>The registered termination function should not contain endless loops or functions that might block the application.</p>						
Parameters	<p>pHandler Specifies the pointer to the handler of the function to be registered. The handler function must be of <code>SrtkTApp_Handler</code> type, i.e. <code>void MyBackgroundHandler(void)</code>.</p>						
Return value	<p>This function returns an error code.</p> <table border="1"> <thead> <tr> <th>Symbol</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>SRTK_ERROR</td> <td>The registration failed.</td> </tr> <tr> <td>SRTK_NO_ERROR</td> <td>The registration was performed without error.</td> </tr> </tbody> </table>	Symbol	Meaning	SRTK_ERROR	The registration failed.	SRTK_NO_ERROR	The registration was performed without error.
Symbol	Meaning						
SRTK_ERROR	The registration failed.						
SRTK_NO_ERROR	The registration was performed without error.						

Related topics	References				
	<table> <tr> <td>RTLIB_REGISTER_BACKGROUND_HANDLER.....</td> <td>190</td> </tr> <tr> <td>RTLIB_REGISTER_TERMINATE_HANDLER.....</td> <td>191</td> </tr> </table>	RTLIB_REGISTER_BACKGROUND_HANDLER	190	RTLIB_REGISTER_TERMINATE_HANDLER	191
RTLIB_REGISTER_BACKGROUND_HANDLER	190				
RTLIB_REGISTER_TERMINATE_HANDLER	191				

RTLIB_MALLOC_PROT

Purpose	To allocate memory with protection against interrupts by using the <code>malloc</code> routine of the standard C library.
	<div style="border: 1px solid #ccc; padding: 5px;"> <p>Tip</p> <p>This macro is provided for backward compatibility only. On the MicroLabBox, use the routines for memory allocation (<code>malloc</code>, <code>calloc</code>, <code>realloc</code> and <code>free</code>).</p> </div>

Syntax	<code>RTLIB_MALLOC_PROT(void *pointer, UInt32 size)</code>
Include file	SrtkStd.h

Parameters	pointer Specifies the address of the allocated buffer.
	size Specifies the memory size to be allocated.

Related topics	References						
	<table> <tr> <td>RTLIB_CALLOC_PROT.....</td> <td>193</td> </tr> <tr> <td>RTLIB_FREE_PROT.....</td> <td>194</td> </tr> <tr> <td>RTLIB_REALLOC_PROT.....</td> <td>194</td> </tr> </table>	RTLIB_CALLOC_PROT.....	193	RTLIB_FREE_PROT.....	194	RTLIB_REALLOC_PROT.....	194
RTLIB_CALLOC_PROT.....	193						
RTLIB_FREE_PROT.....	194						
RTLIB_REALLOC_PROT.....	194						

RTLIB_CALLOC_PROT

Purpose	To allocate memory for an array with protection against interrupts by using the <code>calloc</code> routine of the standard C library.
----------------	--

Tip

This macro is provided for backward compatibility only. On the MicroLabBox, use the routines for memory allocation (`malloc`, `calloc`, `realloc` and `free`).

Syntax	<code>RTLIB_CALLOC_PROT(void *pointer, UInt32 nobj, UInt32 size)</code>
---------------	---

Include file	<code>SrtkStd.h</code>
---------------------	------------------------

Parameters	pointer Specifies the address of the allocated buffer.
	nobj Specifies the number of elements.
	size Specifies the size of one element.

Related topics	References						
	<table> <tr> <td>RTLIB_FREE_PROT.....</td> <td>194</td> </tr> <tr> <td>RTLIB_MALLOC_PROT.....</td> <td>192</td> </tr> <tr> <td>RTLIB_REALLOC_PROT.....</td> <td>194</td> </tr> </table>	RTLIB_FREE_PROT.....	194	RTLIB_MALLOC_PROT.....	192	RTLIB_REALLOC_PROT.....	194
RTLIB_FREE_PROT.....	194						
RTLIB_MALLOC_PROT.....	192						
RTLIB_REALLOC_PROT.....	194						

RTLIB_REALLOC_PROT

Purpose

To change the memory size with protection against interrupts by using the `realloc` routine of the standard C library.

Tip

This macro is provided for backward compatibility only. On the MicroLabBox, use the routines for memory allocation (`malloc`, `calloc`, `realloc` and `free`).

Syntax

```
RTLIB_REALLOC_PROT(void *pointer, UInt32 size)
```

Include file

`SrtkStd.h`

Parameters

pointer Specifies the address of the allocated buffer.
size Specifies the memory size to be allocated.

Related topics

References

RTLIB_CALLOC_PROT.....	193
RTLIB_FREE_PROT.....	194
RTLIB_MALLOC_PROT.....	192

RTLIB_FREE_PROT

Purpose

To free the allocated memory with protection against interrupts by using the `free` routine of the standard C library.

Tip

This macro is provided for backward compatibility only. On the MicroLabBox, use the routines for memory allocation (`malloc`, `calloc`, `realloc` and `free`).

Syntax

```
RTLIB_FREE_PROT(void *pointer)
```

Include file SrtkStd.h

Parameters **pointer** Specifies the address of the buffer to be freed.

Related topics **References**

RTLIB_CALLOC_PROT.....	193
RTLIB_MALLOC_PROT.....	192
RTLIB_REALLOC_PROT.....	194

I/O Functions

Introduction

The board's RTLib provides I/O functions for generating and measuring analog and digital signals.

Where to go from here

Information in this section

Common Concepts for Implementing I/O Functions.....	198
A/D Conversion.....	202
D/A Conversion.....	259
Bit I/O.....	268
Timing I/O.....	317

Common Concepts for Implementing I/O Functions

Introduction

All the I/O driver objects provide similar functions that are to be used for creating, configuring, and starting a functionality on an I/O channel of the board.

These concepts are also used for the board's system functions:

- [Buzzer Control](#) on page 153
- [LED Control](#) on page 162
- [Sensor Supply](#) on page 171

Implementing I/O Functions

Introduction

If you want to implement an I/O function for MicroLabBox, the structure of your source code is similar for each I/O function.

Creating an I/O driver object

First, you must create an I/O driver object representing the required functionality. This driver object provides a default configuration. For standard use cases, you can use it without changes. For information on the initial values, see description of the specific `create` function.

When creating the driver object, you must also specify the channel(s) you want to use. You can access the I/O channels then only via the related driver object. The driver object therefore contains a resource management that checks the availability of the specified channel(s). Each channel can only be assigned once. This is especially relevant for the digital I/O channels. These are bidirectional digital channels used for digital input and digital output features. If you have assigned one digital channel to a driver object, this channel is not available for the other digital features.

The driver object also stores data that you read from an input channel or that you want to write to an output channel.

```
xxxxSDrvObj* MyDriverObject;  
IErr=xxxx_create(&MyDriverObject, Parameters);
```

Changing the initial configuration of the driver object

General settings of an I/O functionality should be set during the initialization phase of the real-time application. Via the `set` functions, you can change the default configuration. Whether a set function can also be used during run time of the real-time application is described in the specific function description.

```
...  
IErr=xxxx_set<ConfigurationSetting>(MyDriverObject, Value);
```

See the function description of your I/O feature to get information on the configuration settings that you can change.

Applying settings to the driver object

The default settings and the custom settings that you specified via the `set` functions must be applied to the driver object at the end of the initialization phase. Call the related `apply` function to do so.

You must call this function also to apply the default configuration to the driver object.

```
...
IErr=xxxx_apply(MyDriverObject);
```

Some I/O functions can be used to generate interrupts or trigger signals on specific conditions. This must also be configured during the initialization phase.

Generating interrupts and trigger signals

If an I/O feature provides a relevant event, for example, for starting a function or reading data, this event can be made available for the generation of interrupts and trigger signals. Interrupts and trigger signals must be initialized separately by using the `setInterruptMode` and `setTriggerMode` functions. If there are configuration settings for downsampling or delaying the signal generation for example, they apply to both, interrupts and trigger signals.

Interrupts Interrupts are used to internally control the real-time application, for example, to start an interrupt service routine that you specified by the driver-specific `setIoIntVector` function.

You must explicitly enable the interrupt generation by using the driver-specific `enableInterrupts` function. You can do this during run time of the real-time application. To stop the interrupt generation, use the driver-specific `disableInterrupts` function.

Note

An enabled interrupt is generated only if the related channel has already been started, refer to [Activating the I/O channels](#) on page 200.

```

...
IErr=xxx_setInterruptMode(MyDriverObject, InterruptType);
# Register the interrupt handler function
IErr=xxx_setIoIntVector(MyDriverObject, InterruptType,
MyInterruptFunction);
# Set the delay value for interrupts (and trigger signals)
IErr=xxx_setEventDelay(MyDriverObject, Value);
# Set the downsample value for interrupts (and trigger signals)
IErr=xxx_setEventDownSample(MyDriverObject, Value);
# Apply the settings
IErr=xxxx_apply(MyDriverObject);
# Start the related channel
IErr=xxx_start(MyDriverObject);
...
# Enable interrupt generation
IErr=xxx_enableInterrupts(MyDriverObject, InterruptType);
...
# Disable interrupt generation
IErr=xxx_disableInterrupts(MyDriverObject, InterruptType);
...

```

Trigger signals Trigger signals are transmitted via a trigger line. A trigger line can be connected internally to another function that consumes the trigger signal, or externally to a digital output channel for controlling any external device. To use a trigger signal internally, use `setTriggerLineIn` so that the consuming function can specify which trigger line it has to listen to.

In contrast to interrupts, you do not need to enable a trigger line explicitly. The trigger signals are automatically enabled after you have activated the related channel by calling the `start` function, refer to [Activating the I/O channels](#) on page 200.

```

...
# Set the trigger mode
IErr=xxx_setTriggerMode(MyDriverObject, TriggerType);
# Set the trigger line to be used for transmitting the generated
trigger signal
IErr=xxx_setTriggerLineOut(MyDriverObject, TriggerLineNumber);
# Set the delay value for the trigger signals (and interrupts)
IErr=xxx_setEventDelay(MyDriverObject, Value);
# Set the downsample value for trigger signals (and interrupts)
IErr=xxx_setEventDownSample(MyDriverObject, Value);
# Apply the settings
IErr=xxxx_apply(MyDriverObject);
# Start the related channel
IErr=xxx_start(MyDriverObject);

```

See the function description of your I/O feature to get information on whether it supports the generation of interrupts and trigger signals.

Activating the I/O channels

The I/O channels are initially disabled at startup or after a reset of the board. To use the channels for input and output signals, you must activate them by using the `start` function. This also enables the generation of trigger signals.

Until you execute the **start** function for your driver object, input channels read zeros and output channels are in high impedance state.

```
...  
IErr=xxxx_start(MyDriverObject);
```

When the channels are not started, the generation of interrupts and trigger signals is also disabled.

Writing settings or values during run time

Some of the **set** functions are intended to be called during run time of the real-time application. To apply these settings or writing the specified value to the I/O channel, you have to call the **write** function for the driver object.

```
...  
IErr=xxxx_set<Parameter>(MyDriverObject, Value);  
IErr=xxxx_write(MyDriverObject);
```

See the function description of your I/O feature to get information on the settings and values that you can change during run time.

Reading values during run time

If you are using an I/O function that gets a setting or a value, you must first call the **read** function for the driver object to store the channel data to the internal buffer of the object. Then, you must call a specific **get** function to get access to the value in your real-time application.

```
...  
IErr=xxxx_read(MyDriverObject);  
IErr=xxx_get<Parameter>(MyDriverObject, &Value);
```

A/D Conversion

Introduction	MicroLabBox provides two different units for A/D conversion. For basic information on MicroLabBox's ADC units, refer to A/D Conversion (MicroLabBox Features) .
---------------------	--

Where to go from here	Information in this section
	Analog In Class 1 202 Analog In Class 2 250
	Information in other sections
	Implementing I/O Functions 198

Analog In Class 1

Introduction	The Analog In Class 1 unit can be accessed via the AdcCl1AnalogIn functions.
---------------------	--

Where to go from here	Information in this section
	AdcCl1AnalogIn_abortBurst 204 To stop the current burst of conversions for one ADC class 1 channel. AdcCl1AnalogIn_abortBurstMultiple 205 To stop the current burst of conversions for multiple ADC class 1 channels. AdcCl1AnalogIn_apply 206 To apply the initialization settings to the ADC Class 1 channel. AdcCl1AnalogIn_create 208 To create the I/O driver object for analog signal inputs via the ADC Class 1 unit. AdcCl1AnalogIn_disableInterrupts 209 To disable the generation of interrupts. AdcCl1AnalogIn_enableInterrupts 211 To enable the generation of interrupts.

AdcCl1AnalogIn_getBurstCurrent	212
To get the current values of a burst conversion.	
AdcCl1AnalogIn_getBurstImmediate	214
To immediately get the values of a burst conversion.	
AdcCl1AnalogIn_getFailureInfo	216
To get the failure information for an ADC Class 1 channel.	
AdcCl1AnalogIn_getSingleValue	218
To get a single value from the read buffer.	
AdcCl1AnalogIn_isDataReady	219
To query the data ready flag of an ADC Class 1 channel.	
AdcCl1AnalogIn_read	221
To read the measured value(s) from an ADC Class 1 channel.	
AdcCl1AnalogIn_readFailureInfo	222
To read the failure information for an ADC Class 1 channel.	
AdcCl1AnalogIn_setBurstMode	223
To set the sample mode for burst conversions.	
AdcCl1AnalogIn_setBurstSize	225
To set the burst size of the specified ADC Class 1 channel.	
AdcCl1AnalogIn_setBurstTrigger	226
To set the burst trigger source.	
AdcCl1AnalogIn_setChannelState	228
To set the state of an ADC Class 1 channel.	
AdcCl1AnalogIn_setConversionMode	229
To set the conversion mode of the specified ADC Class 1 channel.	
AdcCl1AnalogIn_setConversTrigger	231
To set the conversion trigger source.	
AdcCl1AnalogIn_setInterruptMode	233
To set the interrupt mode of the ADC Class 1 channel.	
AdcCl1AnalogIn_setIntVector	234
To set the interrupt handler for processing interrupts from an ADC Class 1 channel.	
AdcCl1AnalogIn_setTimerPeriod	236
To set the period of the timer of an ADC Class 1 channel.	
AdcCl1AnalogIn_setTriggerLineIn	237
To set the trigger line used as trigger source for an ADC Class 1 channel.	
AdcCl1AnalogIn_setTriggerLineOut	238
To set the trigger line used for transmitting a generated trigger signal.	
AdcCl1AnalogIn_setTriggerLineOutStatus	240
To set the status of the trigger line.	

AdcCl1AnalogIn_setTriggerMode	241
To set the trigger mode of the ADC Class 1 channel.	
AdcCl1AnalogIn_start	243
To enable the driver object to react on a conversion trigger.	
AdcCl1AnalogIn_stop	244
To disable the driver object to react on a conversion trigger.	
AdcCl1AnalogIn_triggerBurst	245
To generate a burst trigger for one ADC class 1 channel.	
AdcCl1AnalogIn_triggerBurstMultiple	246
To generate a burst trigger for multiple ADC class 1 channels.	
AdcCl1AnalogIn_triggerConversion	247
To generate a conversion trigger for one ADC class 1 channel.	
AdcCl1AnalogIn_triggerConversionMultiple	248
To generate a conversion trigger for multiple ADC class 1 channels.	
AdcCl1AnalogIn_write	249
To update settings of the specified ADC Class 1 channel during run time.	

Information in other sections[ADC Class 1 \(MicroLabBox Features\)](#)

AdcCl1AnalogIn_abortBurst

Syntax

```
UInt32 AdcCl1AnalogIn_abortBurst(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn)
```

Include file

IoDrvAdcClass1AnalogIn.h

Purpose

To stop the current burst of conversions for one ADC class 1 channel.

Description

If you use this function to stop a burst of conversions, the conversion results are immediately available in the read buffer, even if the write buffer has not been completely filled. You can read the values using [AdcCl1AnalogIn_read](#).

In continuous burst sample mode, you only stop the current burst of conversions. The next burst is started automatically.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using [AdcCl1AnalogIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_setConversionMode.....	229
AdcCl1AnalogIn_triggerBurstMultiple.....	246
AdcCl1AnalogIn_write.....	249

AdcCl1AnalogIn_abortBurstMultiple

Syntax

```
UInt32 AdcCl1AnalogIn_abortBurstMultiple(UInt32 ChannelMask)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To stop the current burst of conversions for multiple ADC class 1 channels.

Description

If you use this function to stop the burst of conversions for several ADC class 1 channels, the associated channels must be specified in a channel mask. The

conversion results are immediately available in the read buffer, even if the write buffer has not been completely filled. You can read the values using [AdcCl1AnalogIn_read](#).

In continuous burst sample mode, you only stop the current burst of conversions. The next burst is started automatically.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [ADC Class 1](#) ([MicroLabBox Features](#)).

Parameters

ChannelMask Lets you specify which channels in the range 1 ... 16 will stop their current burst of conversions. You can combine the predefined symbols by using the bitwise OR operator.

Symbol	Meaning
ADC_CLASS1_MASK_CH_1	Specifies channel 1.
...	...
ADC_CLASS1_MASK_CH_16	Specifies channel 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_create	208
AdcCl1AnalogIn_setConversionMode	229
AdcCl1AnalogIn_write	249

[AdcCl1AnalogIn_apply](#)**Syntax**

```
UInt32 AdcCl1AnalogIn_apply(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn)
```

Include file	<code>IoDrvAdcClass1AnalogIn.h</code>						
Purpose	To apply the initialization settings to the ADC class 1 channel.						
Description	<p>Before you can start the specified A/D converter, you must transfer the settings of the <code>AdcCl1AnalogIn</code> driver object to its related input channels by using <code>AdcCl1AnalogIn_apply</code>. You must do this also for the default values and for values that you specified by using the <code>AdcCl1AnalogIn_set<Parameter></code> functions. The <code>AdcCl1AnalogIn_apply</code> function is intended to be called at the end of the initialization phase of the real-time application.</p> <p>The signal measurement on the specified analog input channel is not activated until you have called <code>AdcCl1AnalogIn_start</code>.</p> <p>To update settings during run time, use <code>AdcCl1AnalogIn_write</code>.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features).</p>						
Parameters	<p>pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The <code>AdcCl1AnalogIn</code> driver object must already have been created by using <code>AdcCl1AnalogIn_create</code>.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td> <td>The function was successfully completed.</td> </tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						
Related topics	<p>References</p> <p>AdcCl1AnalogIn_create..... 208</p>						

AdcCl1AnalogIn_create

Syntax	<pre>UInt32 AdcCl1AnalogIn_create(AdcCl1AnalogInSDrvObject **ppAdcCl1AnalogIn, UInt32 Channel)</pre>
Include file	<code>IoDrvAdcClass1AnalogIn.h</code>
Purpose	To create the I/O driver object for analog signal inputs via the ADC class 1 unit.
Description	<p>This function performs all the steps necessary to create an I/O driver object for the ADC class 1 unit. It specifies one analog input channel for signal measurement. The instantiated ADC class 1 converter can be flexibly configured, for example, it can be operated in single or burst conversion mode.</p> <p>The ADC class 1 driver object is initialized with default values so that it is ready for use.</p> <p>The initial values of the ADC class 1 driver object are:</p> <ul style="list-style-type: none"> ▪ Conversion mode: single conversion mode ▪ Burst trigger: software (implicit in single conversion mode) ▪ Conversion trigger: timer ▪ Timer period: 100 µs ▪ Burst sample mode: continuous mode ▪ Burst size: one conversion (implicit in single conversion mode) ▪ Time stamp mode: disabled ▪ Interrupt mode: no interrupt generation ▪ Trigger mode: no trigger signal generation ▪ Trigger line: no trigger line configured ▪ Event downsample: 1 ▪ Event delay: 0 s <p>The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features).</p>

Parameters

ppAdcCl1AnalogIn Lets you specify the address of a variable which is to hold the address of the created driver object.

Channel Lets you specify the channel to be accessed by this driver in the range 1 ... 24.

Symbol	Meaning
ADC_CLASS1_CHANNEL_1	Specifies channel 1 of the ADC class 1 unit.
...	...
ADC_CLASS1_CHANNEL_24	Specifies channel 24 of the ADC class 1 unit.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example

This example shows how to create an I/O driver object for channel 4 of the ADC class 1 unit:

```
UInt32 IError;
AdcCl1AnalogInSDrvObject *MyAnalogInDrv;
IError = AdcCl1AnalogIn_create(&pMyAnalogInDrv,
ADC_CLASS1_CHANNEL_4);
```

Related topics**References**

[AdcCl1AnalogIn_apply](#)..... 206

AdcCl1AnalogIn_disableInterrupts

Syntax

```
UInt32 AdcCl1AnalogIn_disableInterrupts(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 InterruptSelect)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To disable the generation of interrupts.

Description	<p>With this function, you can disable the generation of interrupts that you specified by using <code>AdcCl1AnalogIn_setInterruptMode</code>. The settings take effect immediately.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>								
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features).</p>								
Parameters	<p>pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using <code>AdcCl1AnalogIn_create</code>.</p> <p>InterruptSelect Lets you specify the interrupts that are to be disabled. The predefined symbols can be combined by using the bitwise OR operator.</p>								
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Symbol</th> <th style="text-align: left; padding: 2px;">Meaning</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">ADC_CLASS1_INT_BURST_CONV_START</td> <td style="padding: 2px;">Disables an interrupt generated on <ul style="list-style-type: none"> ▪ Conversion start (single conversion mode) ▪ Burst start (burst conversion mode) </td> </tr> <tr> <td style="padding: 2px;">ADC_CLASS1_INT_BURST_CONV_END</td> <td style="padding: 2px;">Disables an interrupt generated on <ul style="list-style-type: none"> ▪ Conversion end (single conversion mode) ▪ Burst end (burst conversion mode) </td> </tr> <tr> <td style="padding: 2px;">ADC_CLASS1_INT_CHANNEL_FAILURE</td> <td style="padding: 2px;">Disables an interrupt generated on channel failure.</td> </tr> </tbody> </table>	Symbol	Meaning	ADC_CLASS1_INT_BURST_CONV_START	Disables an interrupt generated on <ul style="list-style-type: none"> ▪ Conversion start (single conversion mode) ▪ Burst start (burst conversion mode) 	ADC_CLASS1_INT_BURST_CONV_END	Disables an interrupt generated on <ul style="list-style-type: none"> ▪ Conversion end (single conversion mode) ▪ Burst end (burst conversion mode) 	ADC_CLASS1_INT_CHANNEL_FAILURE	Disables an interrupt generated on channel failure.	
Symbol	Meaning								
ADC_CLASS1_INT_BURST_CONV_START	Disables an interrupt generated on <ul style="list-style-type: none"> ▪ Conversion start (single conversion mode) ▪ Burst start (burst conversion mode) 								
ADC_CLASS1_INT_BURST_CONV_END	Disables an interrupt generated on <ul style="list-style-type: none"> ▪ Conversion end (single conversion mode) ▪ Burst end (burst conversion mode) 								
ADC_CLASS1_INT_CHANNEL_FAILURE	Disables an interrupt generated on channel failure.								
Return value	The function returns an error code.								
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Error Code</th> <th style="text-align: left; padding: 2px;">Meaning</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">IOLIB_NO_ERROR</td> <td style="padding: 2px;">The function was successfully completed.</td> </tr> <tr> <td style="padding: 2px;">!= IOLIB_NO_ERROR</td> <td style="padding: 2px;">The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.			
Error Code	Meaning								
IOLIB_NO_ERROR	The function was successfully completed.								
!= IOLIB_NO_ERROR	The function was not completed.								
Related topics	References								
	<table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="padding: 2px; vertical-align: bottom;"> AdcCl1AnalogIn_create.....</td> <td style="padding: 2px; vertical-align: bottom;">208</td> </tr> <tr> <td style="padding: 2px; vertical-align: bottom;"> AdcCl1AnalogIn_enableInterrupts.....</td> <td style="padding: 2px; vertical-align: bottom;">211</td> </tr> <tr> <td style="padding: 2px; vertical-align: bottom;"> AdcCl1AnalogIn_setInterruptMode.....</td> <td style="padding: 2px; vertical-align: bottom;">233</td> </tr> </tbody> </table>	AdcCl1AnalogIn_create	208	AdcCl1AnalogIn_enableInterrupts	211	AdcCl1AnalogIn_setInterruptMode	233		
AdcCl1AnalogIn_create	208								
AdcCl1AnalogIn_enableInterrupts	211								
AdcCl1AnalogIn_setInterruptMode	233								

AdcCl1AnalogIn_enableInterrupts

Syntax

```
UInt32 AdcCl1AnalogIn_enableInterrupts(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 InterruptSelect)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To enable the generation of interrupts.

Description

With this function, you can either enable the generation of interrupts or reenable the generation of interrupts that you disabled by using `AdcCl1AnalogIn_disableInterrupts`. The interrupts must already have been specified by using `AdcCl1AnalogIn_setInterruptMode`. The settings take effect immediately.

After calling `AdcCl1AnalogIn_start`, interrupts are disabled.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using `AdcCl1AnalogIn_create`.

InterruptSelect Lets you specify the interrupts that are to be enabled. The predefined symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
<code>ADC_CLASS1_INT_BURST_CONV_START</code>	Enables an interrupt generated on <ul style="list-style-type: none"> ▪ Conversion start (single conversion mode) ▪ Burst start (burst conversion mode)
<code>ADC_CLASS1_INT_BURST_CONV_END</code>	Enables an interrupt generated on <ul style="list-style-type: none"> ▪ Conversion end (single conversion mode) ▪ Burst end (burst conversion mode)
<code>ADC_CLASS1_INT_CHANNEL_FAILURE</code>	Enables an interrupt generated on channel failure.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References						
	<table> <tr> <td>AdcCl1AnalogIn_create.....</td> <td>208</td> </tr> <tr> <td>AdcCl1AnalogIn_disableInterrupts.....</td> <td>209</td> </tr> <tr> <td>AdcCl1AnalogIn_setInterruptMode.....</td> <td>233</td> </tr> </table>	AdcCl1AnalogIn_create.....	208	AdcCl1AnalogIn_disableInterrupts.....	209	AdcCl1AnalogIn_setInterruptMode.....	233
AdcCl1AnalogIn_create.....	208						
AdcCl1AnalogIn_disableInterrupts.....	209						
AdcCl1AnalogIn_setInterruptMode.....	233						

AdcCl1AnalogIn_getBurstCurrent

Syntax	<pre>UInt32 AdcCl1AnalogIn_getBurstCurrent(AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn, UInt32 Offset, UInt32 Count, UInt32 *pIsNew, Float64 *pValues)</pre>
Include file	IoDrvAdcClass1AnalogIn.h
Purpose	To get the current values of a burst conversion.
Description	<p>If you have specified to operate in burst conversion mode, you can use this function to read the conversion results of a burst from the internal buffer. Optionally, you can read only a segment of the conversion results by specifying a start address (offset) and the number of conversion results to be read (count). The first conversion result of the segment is stored in the first element of the data array specified by the pValues parameter.</p> <p>To update the internal buffer with the latest conversion results, you must execute AdcCl1AnalogIn_read beforehand. However, as long as the current burst conversion is not completed, the read values come from the preceding burst conversion. The pIsNew parameter is used to indicate whether the values are the newest ones (the values were not read before) or ones from the preceding burst conversion (the old values are read again).</p> <p>The function is intended to be called during run time of the real-time application.</p>

If an error is detected, the very first instance of the error is returned to the caller.

Note

- The length of the data array specified by the `pValues` parameter must be at least the number of elements specified by the `Count` parameter.
- The sum of the `Offset` and `Count` parameters must not exceed the burst size specified by using `AdcCl1AnalogIn_setBurstSize`.
- All analog inputs of the ADC class 1 unit are in a disabled state after reset. To enable the inputs accessed by this driver, use `AdcCl1AnalogIn_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

`pAdcCl1AnalogIn` Lets you specify the ADC Class 1 channel to be used. The `AdcCl1AnalogIn` driver object must already have been created by using `AdcCl1AnalogIn_create`.

`Offset` Lets you specify the address in the conversion result from which values are delivered to the `pValues` parameter. You can specify a value in the range 0 ... (burst size - 1).

To deliver the complete conversion result, you have to specify an offset of 0 or `ADC_CLASS1_COMPLETE_BUFFER` as `Count` parameter.

`Count` Lets you specify the number of conversion results to be delivered in the range 1 ... (burst size - `Offset`). If the specified number exceeds the current burst size, the number of results to be delivered is automatically set to the current burst size.

You can use the following symbol to specify the complete buffer to be delivered independently of the specified size of the current burst.

Symbol	Meaning
<code>ADC_CLASS1_COMPLETE_BUFFER</code>	The complete buffer is to be delivered.

`plsNew` Lets you specify the address of a variable that holds a flag indicating whether the measured values are the newest ones.

Flag Value	Meaning
0	The current values come from a preceding burst conversion.
> 0	The current values come from the newest burst conversion.

pValues Lets you specify the address of a variable that holds the measured values in the range -1.0 ... +1.0 with a resolution of 16 bits.

Value	Voltage
-1.0	-10.0 V
...	...
0.0	0.0 V
...	...
+1.0	+10 V

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_getBurstImmediate.....	214
AdcCl1AnalogIn_read.....	221
AdcCl1AnalogIn_setBurstSize.....	225
AdcCl1AnalogIn_start.....	243

AdcCl1AnalogIn_getBurstImmediate

Syntax

```
UInt32 AdcCl1AnalogIn_getBurstImmediate(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 *pCount,
    Float64 *pValues)
```

Include file

IoDrvAdcClass1AnalogIn.h

Purpose

To immediately get the values of a burst conversion.

Description

This function must be used if a running burst of conversions was stopped by using **AdcCl1AnalogIn_abortBurst** or **AdcCl1AnalogIn_abortBurstMultiple**. You do not have to wait for

completion of the burst. However, the number of available conversion results might differ from the specified burst size. The function therefore outputs the number of currently available conversion results.

This function gets the measuring values of the voltage of the analog input signal sampled by a burst of conversions on a channel of the ADC class 1 unit. The analog input channel to be evaluated is selected by specifying the I/O driver object that is associated with the target input channel. The function does not directly transfer the requested values from the real hardware but retrieves them from internal information of the driver. To update the internal information with the latest measuring values, the function `AdcCl1AnalogIn_read()` must be executed beforehand.

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

The length of the data array specified by the `pValues` parameter must be large enough so that it can hold at least as many values as specified by the burst size.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The `AdcCl1AnalogIn` driver object must already have been created by using `AdcCl1AnalogIn_create`.

pCount Lets you specify the address of a variable that holds the current number of conversion results contained in the `pValues` parameter.

pValues Lets you specify the address of a variable that holds the measured values in the range -1.0 ... +1.0 with a resolution of 16 bits.

Value	Voltage
-1.0	-10.0 V
...	...
0.0	0.0 V
...	...
+1.0	+10 V

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References						
	<table> <tr> <td>AdcCl1AnalogIn_abortBurst.....</td> <td>204</td> </tr> <tr> <td>AdcCl1AnalogIn_setBurstSize.....</td> <td>225</td> </tr> <tr> <td>AdcCl1AnalogIn_start.....</td> <td>243</td> </tr> </table>	AdcCl1AnalogIn_abortBurst	204	AdcCl1AnalogIn_setBurstSize	225	AdcCl1AnalogIn_start	243
AdcCl1AnalogIn_abortBurst	204						
AdcCl1AnalogIn_setBurstSize	225						
AdcCl1AnalogIn_start	243						

AdcCl1AnalogIn_getFailureInfo

Syntax	<pre>UInt32 AdcCl1AnalogIn_getFailureInfo(AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn, UInt32 *pChannelFailures)</pre>
Include file	<code>IoDrvAdcClass1AnalogIn.h</code>
Purpose	To get the failure information for an ADC class 1 channel.
Description	<p>After you have read the failure information from the specified channel and stored it internally to the AdcCl1AnalogIn driver object by using AdcCl1AnalogIn_readFailureInfo, you can use this function to get the contents of the failure information.</p> <p>The following failure types are defined:</p> <ul style="list-style-type: none"> ▪ No burst start Indicates that a burst trigger was missing. The start of a conversion within a burst was requested but the burst was not started. ▪ Burst trigger overflow Indicates that a burst trigger overflow occurred. Burst trigger overflow means that a burst start was requested while the previous burst was still in progress. ▪ Conversion trigger overflow Indicates that a conversion trigger overflow occurred. Conversion trigger overflow means that a conversion start was requested while the previous conversion was still in progress.

- Store error

Indicates that a store error occurred. Store error means that an error occurred while saving the value to the buffer.

- Data lost

Indicates that measured values were lost. Data lost means that no free buffer for writing conversion results was available because the data had not been read at all or not fast enough.

The function is intended to be called in the function that handles the interrupt generated on a channel failure to evaluate the reason for the interrupt.

Nevertheless, the function can be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using [AdcCl1AnalogIn_create](#).

pChannelFailures Lets you specify the address of a variable that holds the failure information of the specified ADC class 1 channel. The failure information contains one or multiple of the following symbols (combined via the bitwise OR operator).

Symbol	Meaning
ADC_CLASS1_NO_BURST_START	Conversion start was requested at the same time or before the burst was started.
ADC_CLASS1_BURST_TRIGGER_OVFL	Burst start was requested while previous burst was still in progress.
ADC_CLASS1_CONV_TRIGGER_OVFL	Conversion start was requested while previous conversion was still in progress.
ADC_CLASS1_DATA_LOST	Measuring data was lost because free buffer for writing conversion results was not available.
ADC_CLASS1_STORE_ERROR	Could not store measuring values.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References						
	<table> <tr> <td>AdcCl1AnalogIn_create.....</td> <td>208</td> </tr> <tr> <td>AdcCl1AnalogIn_readFailureInfo.....</td> <td>222</td> </tr> <tr> <td>AdcCl1AnalogIn_setInterruptMode.....</td> <td>233</td> </tr> </table>	AdcCl1AnalogIn_create.....	208	AdcCl1AnalogIn_readFailureInfo.....	222	AdcCl1AnalogIn_setInterruptMode.....	233
AdcCl1AnalogIn_create.....	208						
AdcCl1AnalogIn_readFailureInfo.....	222						
AdcCl1AnalogIn_setInterruptMode.....	233						

AdcCl1AnalogIn_getSingleValue

Syntax	<pre>UInt32 AdcCl1AnalogIn_getSingleValue(AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn, UInt32 *pIsNew, Float64 *pValue)</pre>
Include file	<code>IoDrvAdcClass1AnalogIn.h</code>
Purpose	To get a single value from the read buffer.
Description	<p>Because you get the value from the internal buffer of the driver object, there might be a newer value if a conversion is in progress. To update the internal buffer with the latest value, you have to call <code>AdcCl1AnalogIn_read</code> beforehand. However, until the current conversion is completed, the read value comes from the preceding conversion. The <code>pIsNew</code> parameter is used to indicate whether the value is the newest one (the value was not read before) or one from a preceding conversion (the old value is read again).</p> <p>The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.</p>
I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) .

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using [AdcCl1AnalogIn_create](#).

plsNew Lets you specify the address of a variable that holds a flag indicating whether the measured value is the newest one.

Flag Value	Meaning
0	The current value comes from a preceding conversion.
> 0	The current value comes from the newest conversion.

pValue Lets you specify the address of a variable that holds the measured value in the range -1.0 ... +1.0 with a resolution of 16 bits.

Value	Voltage
-1.0	-10.0 V
...	...
0.0	0.0 V
...	...
+1.0	+10 V

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_create	208
AdcCl1AnalogIn_isDataReady	219
AdcCl1AnalogIn_read	221

[AdcCl1AnalogIn_isDataReady](#)**Syntax**

```
UInt32 AdcCl1AnalogIn_isDataReady(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 *pIsDataReady)
```

Include file	<code>IoDrvAdcClass1AnalogIn.h</code>						
Purpose	To query the data ready flag of an ADC class 1 channel.						
Description	<p>In single conversion mode, new data can be read upon completion of a single conversion. In burst conversion mode, new data can be read upon completion of a burst of conversions. The function reads the information directly from the specified ADC class 1 channel. A preceding call of another function to fetch the result is not required.</p> <p>The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features).</p>						
Parameters	<p>pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using AdcCl1AnalogIn_create.</p> <p>plsDataReady Lets you specify the address of a variable that holds the result of the data ready query. The value of the variable means the following:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>0</td><td>New data is not available.</td></tr> <tr> <td>> 0</td><td>New data is available and ready to be read.</td></tr> </tbody> </table>	Value	Meaning	0	New data is not available.	> 0	New data is available and ready to be read.
Value	Meaning						
0	New data is not available.						
> 0	New data is available and ready to be read.						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	References						
	<table border="0"> <tr> <td>AdcCl1AnalogIn_getBurstCurrent.....</td><td>212</td></tr> <tr> <td>AdcCl1AnalogIn_getSingleValue.....</td><td>218</td></tr> <tr> <td>AdcCl1AnalogIn_read.....</td><td>221</td></tr> </table>	AdcCl1AnalogIn_getBurstCurrent.....	212	AdcCl1AnalogIn_getSingleValue.....	218	AdcCl1AnalogIn_read.....	221
AdcCl1AnalogIn_getBurstCurrent.....	212						
AdcCl1AnalogIn_getSingleValue.....	218						
AdcCl1AnalogIn_read.....	221						

AdcCl1AnalogIn_read

Syntax	<code>UInt32 AdcCl1AnalogIn_read(AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn)</code>
--------	--

Include file	<code>IoDrvAdcClass1AnalogIn.h</code>
--------------	---------------------------------------

Purpose	To read the measured value(s) from an ADC class 1 channel.
---------	--

Description	This function reads the raw data from the specified analog input channel and calculates the corresponding conversion result, which is then stored in the internal buffer. The conversion result consists either of one measured value (if you operate in single conversion mode) or of several measured values (if you operate in burst conversion mode).
-------------	---

Depending on the conversion mode used, you can get the value(s) from the internal buffer by using:

- `AdcCl1AnalogIn_getSingleValue`,
- `AdcCl1AnalogIn_getBurstImmediate`
- `AdcCl1AnalogIn_getBurstCurrent`

The `AdcCl1AnalogIn_getxxx` functions (except for `AdcCl1AnalogIn_getBurstImmediate`) provide a flag that indicates whether the values come from the latest conversion or from the preceding one if the current conversion was not completed yet. To ensure that you get the latest conversion results, you can do the following:

- You can use `AdcCl1AnalogIn_isDataReady` to poll the data ready flag indicating that a new conversion result of a single or a burst conversion is available.
- Alternatively, you can implement the handling of the *end of conversion* or *end of burst conversion* interrupt that is raised if a conversion or a burst of conversions was completed. The interrupt can trigger an interrupt service routine including the `AdcCl1AnalogIn_read` function.

Before you can use this function, the analog input channel must be enabled by using `AdcCl1AnalogIn_start`.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) .
-------------	---

	For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features) .
--	--

Parameters **pAdcCl1AnalogIn** Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using [AdcCl1AnalogIn_create](#).

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

AdcCl1AnalogIn_create	208
AdcCl1AnalogIn_getBurstCurrent	212
AdcCl1AnalogIn_getBurstImmediate	214
AdcCl1AnalogIn_getSingleValue	218
AdcCl1AnalogIn_isDataReady	219
AdcCl1AnalogIn_start	243

AdcCl1AnalogIn_readFailureInfo

Syntax

```
UInt32 AdcCl1AnalogIn_readFailureInfo(AdcCl1AnalogInSDrvObject
*pAdcCl1AnalogIn)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To read the failure information for an ADC class 1 channel.

Description

The function reads the failure information of the specified analog input channel and stores it internally in the AdcCl1AnalogIn driver object. To get the failure information, you must use the function [AdcCl1AnalogIn_getFailureInfo](#). For a description of the possible failures, refer to [AdcCl1AnalogIn_getFailureInfo](#) on page 216.

The function is intended to be called in the function that handles the interrupt generated on a channel failure to evaluate the reason for the interrupt. Nevertheless, the function can be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

If you execute this function, the failure information on the channel is cleared.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using [AdcCl1AnalogIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics
References

AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_getFailureInfo.....	216
AdcCl1AnalogIn_setInterruptMode.....	233

AdcCl1AnalogIn_setBurstMode

Syntax

```
UInt32 AdcCl1AnalogIn_setBurstMode(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 BurstSampleMode)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To set the sample mode for burst conversions.

Description	<p>If you have specified to operate in burst conversion mode, you also have to specify the sample mode of the burst conversion.</p> <ul style="list-style-type: none"> ▪ Triggered burst sample mode: A burst of conversions is started once after receiving the burst trigger from the specified burst trigger source. The burst is finished when the specified number of A/D conversions was executed. The next burst must be started with a new burst trigger. ▪ Continuous burst sample mode: Bursts of conversions are executed continuously after receiving an initial burst trigger from the specified burst trigger source. Successive bursts are started automatically. When a burst is completed, the next burst is started immediately after the last A/D conversion of the previous burst. No further burst triggers are required. <p>If you use software triggers for bursts and conversions, you must call the following functions:</p> <ul style="list-style-type: none"> ▪ To start a new burst for one ADC Class 1 channel, you must call AdcCl1AnalogIn_triggerBurst once. Within the bursts you must start each conversion by calling AdcCl1AnalogIn_triggerConversion. ▪ To start a new burst for multiple ADC Class 1 channels, you must call AdcCl1AnalogIn_triggerBurstMultiple once. Within the bursts you must start each conversion by calling AdcCl1AnalogIn_triggerConversionMultiple. <p>The value takes effect only after you have called AdcCl1AnalogIn_apply at the end of the initialization phase.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features).</p>						
Parameters	<p>pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using AdcCl1AnalogIn_create.</p> <p>BurstSampleMode Lets you specify the burst sample mode.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>ADC_CLASS1_TRIGGERED_MODE</td><td>Specifies the triggered burst sample mode (default).</td></tr> <tr> <td>ADC_CLASS1_CONTINUOUS_MODE</td><td>Specifies the continuous burst sample mode.</td></tr> </tbody> </table>	Symbol	Meaning	ADC_CLASS1_TRIGGERED_MODE	Specifies the triggered burst sample mode (default).	ADC_CLASS1_CONTINUOUS_MODE	Specifies the continuous burst sample mode.
Symbol	Meaning						
ADC_CLASS1_TRIGGERED_MODE	Specifies the triggered burst sample mode (default).						
ADC_CLASS1_CONTINUOUS_MODE	Specifies the continuous burst sample mode.						

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_triggerBurst.....	245
AdcCl1AnalogIn_triggerBurstMultiple.....	246
AdcCl1AnalogIn_triggerConversion.....	247
AdcCl1AnalogIn_triggerConversionMultiple.....	248
AdcCl1AnalogIn_write.....	249

AdcCl1AnalogIn_setBurstSize

Syntax

```
UInt32 AdcCl1AnalogIn_setBurstSize(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 BurstSize)
```

Include file

IoDrvAdcClass1AnalogIn.h

Purpose

To set the burst size of the specified ADC class 1 channel.

Description

The burst size specifies the number of conversions within a burst. It also defines the sizes of the write, free, and read buffers required for the swinging buffer mechanism.

The function is intended to be called during the initialization phase of the real-time application.

The value does not take effect until you have called **AdcCl1AnalogIn_apply** at the end of the initialization phase.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Note

This setting is applied only if you specified the burst conversion mode with the `AdcCl1AnalogIn_setConversionMode` function.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [ADC Class 1](#) ([MicroLabBox Features](#)).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC class 1 input channel to be used. The AdcCl1AnalogIn driver object had to be created before by using `AdcCl1AnalogIn_create`.

BurstSize Lets you specify the number of conversions in a burst in the range 1 ... 8192. If you specify a value smaller than 1, the burst size is set to 1. If you specify a value greater than 8192, the burst size is set to 8192.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

<code>AdcCl1AnalogIn_apply</code>	206
<code>AdcCl1AnalogIn_create</code>	208
<code>AdcCl1AnalogIn_setConversionMode</code>	229

AdcCl1AnalogIn_setBurstTrigger

Syntax

```
UInt32 AdcCl1AnalogIn_setBurstTrigger(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 BurstTrigSrc)
```

Include file	<code>IoDrvAdcClass1AnalogIn.h</code>						
Purpose	To set the burst trigger source.						
Description	<p>You can specify different trigger sources to start the bursts of conversions on the specified A/D converter.</p> <ul style="list-style-type: none"> ▪ Timer The burst of conversions is started by an internal timer. You must additionally specify the period of the timer by using <code>AdcCl1AnalogIn_setTimerPeriod</code>. ▪ Software The burst of conversions is started by the <code>AdcCl1AnalogIn_triggerBurst</code> function for one channel or by the <code>AdcCl1AnalogIn_triggerBurstMultiple</code> function for multiple channels in your real-time application. ▪ Trigger line The burst of conversions is started by a trigger line. You must additionally specify the generator of the trigger, for example, by using <code>xxxx_setTriggerMode</code> and <code>xxxx_setTriggerLineOut</code> of a digital I/O function, and the consumer of the trigger by using <code>xxxx_setTriggerLineIn</code>. <p>The setting takes effect only after you have called <code>AdcCl1AnalogIn_apply</code> during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features).</p>						
Parameters	<p>pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using <code>AdcCl1AnalogIn_create</code>.</p> <p>BurstTrigSrc Lets you specify the burst trigger source used for the specified A/D converter.</p> <table border="1"> <thead> <tr> <th>Symbol</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>ADC_CLASS1_TRIGGER_SW</code></td> <td>The burst start is triggered by software (default).</td> </tr> <tr> <td><code>ADC_CLASS1_TRIGGER_TIMER</code></td> <td>The burst start is triggered by the channel-specific timer.</td> </tr> </tbody> </table>	Symbol	Meaning	<code>ADC_CLASS1_TRIGGER_SW</code>	The burst start is triggered by software (default).	<code>ADC_CLASS1_TRIGGER_TIMER</code>	The burst start is triggered by the channel-specific timer.
Symbol	Meaning						
<code>ADC_CLASS1_TRIGGER_SW</code>	The burst start is triggered by software (default).						
<code>ADC_CLASS1_TRIGGER_TIMER</code>	The burst start is triggered by the channel-specific timer.						

Symbol	Meaning
ADC_CLASS1_TRIGGER_TRIG_LINE	The burst start is triggered by a trigger line.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_setTimerPeriod.....	236
AdcCl1AnalogIn_setTriggerLineIn.....	237
AdcCl1AnalogIn_triggerBurst.....	245
AdcCl1AnalogIn_triggerBurstMultiple.....	246

AdcCl1AnalogIn_setChannelState

Syntax

```
UInt32 AdcCl1AnalogIn_setChannelState(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 ChannelState)
```

Include file

IoDrvAdcClass1AnalogIn.h

Purpose

To set the state of an ADC class 1 channel.

Description

This function controls the specified analog input channel during run time. You can use it for starting and stopping an A/D conversion, for example, an A/D conversion running in continuous burst conversion mode.

Disabling the operation of a channel means:

- If a burst of conversions is in progress, this burst is completed. New bursts and conversions will not be started.
- The generation of specified interrupts and trigger signals is stopped.
- The I/O pin of the specified analog input channel is disconnected from the ADC unit.

The setting is not updated until you have called `AdcCl1AnalogIn_write` during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using `AdcCl1AnalogIn_create`.

ChannelState Lets you specify the state of the channel controlled by the AdcCl1AnalogIn driver object.

Symbol	Meaning
ADC_CLASS1_CHANNEL_STATE_DISABLE	Disables channel operation (default).
ADC_CLASS1_CHANNEL_STATE_ENABLE	Enables channel operation.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_write.....	249

AdcCl1AnalogIn_setConversionMode

Syntax

```
UInt32 AdcCl1AnalogIn_setConversionMode(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 ConversionMode)
```

Include file	<code>IoDrvAdcClass1AnalogIn.h</code>						
Purpose	To set the conversion mode of the specified ADC class 1 channel.						
Description	<p>The A/D converter of the ADC class 1 unit can be operated in two operation modes:</p> <ul style="list-style-type: none"> ▪ Single conversion mode After receiving a conversion trigger, the A/D converter delivers one measured value which can be read immediately after its conversion time. ▪ Burst conversion mode After receiving a burst trigger, a burst of conversions is started. The A/D converter can deliver up to 8192 values which can be read when the specified number of A/D conversions has been completed. If the burst conversion mode is selected, you have to specify further burst-specific settings, for example, the burst sample mode. <p>The function is intended to be called during the initialization phase of the real-time application. The value does not take effect until you have called AdcCl1AnalogIn_apply.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features).</p>						
Parameters	<p>pAdcCl1AnalogIn Lets you specify the ADC class 1 input channel to be used. The AdcCl1AnalogIn driver object had to be created before by using AdcCl1AnalogIn_create.</p> <p>ConversionMode Lets you specify the conversion mode to be used for the specified analog input channel.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>ADC_CLASS1_SINGLE_CONV_MODE</td><td>Specifies the single conversion mode (default).</td></tr> <tr> <td>ADC_CLASS1_BURST_CONV_MODE</td><td>Specifies the burst conversion mode.</td></tr> </tbody> </table>	Symbol	Meaning	ADC_CLASS1_SINGLE_CONV_MODE	Specifies the single conversion mode (default).	ADC_CLASS1_BURST_CONV_MODE	Specifies the burst conversion mode.
Symbol	Meaning						
ADC_CLASS1_SINGLE_CONV_MODE	Specifies the single conversion mode (default).						
ADC_CLASS1_BURST_CONV_MODE	Specifies the burst conversion mode.						

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_setBurstMode.....	223

AdcCl1AnalogIn_setConversTrigger

Syntax

```
UInt32 AdcCl1AnalogIn_setConversTrigger(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 ConversionTrigSrc)
```

Include file

IoDrvAdcClass1AnalogIn.h

Purpose

To set the conversion trigger source.

Description

You can specify different trigger sources to start the conversion on the specified A/D converter.

- Timer

The conversion is started by an internal timer. You must additionally specify the period of the timer by using **AdcCl1AnalogIn_setTimerPeriod**.

- Software

The conversion is started by the **AdcCl1AnalogIn_triggerConversion** or **AdcCl1AnalogIn_triggerConversionMultiple** functions in your real-time application.

- Trigger line

The conversion is started by a trigger line. You must additionally specify the generator of the trigger by using **xxxx_setTriggerMode** and **xxxx_setTriggerLineOut**, and the consumer of the trigger by using **xxxx_setTriggerLineIn**.

The setting takes effect only after you have called **AdcCl1AnalogIn_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using [AdcCl1AnalogIn_create](#).

ConversionTrigSrc Lets you specify the conversion trigger source used for the specified A/D converter.

Symbol	Meaning
ADC_CLASS1_TRIGGER_SW	The conversion start is triggered by software.
ADC_CLASS1_TRIGGER_TIMER	The conversion start is triggered by the channel-specific timer (default).
ADC_CLASS1_TRIGGER_TRIG_LINE	The conversion start is triggered by a trigger line.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_setTimerPeriod.....	236
AdcCl1AnalogIn_setTriggerLineIn.....	237
AdcCl1AnalogIn_triggerConversion.....	247
AdcCl1AnalogIn_triggerConversionMultiple.....	248

AdcCl1AnalogIn_setInterruptMode

Syntax

```
UInt32 AdcCl1AnalogIn_setInterruptMode(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 InterruptMode)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To set the interrupt mode of the ADC class 1 channel.

Description

The interrupt mode is used for single conversion mode and burst conversion mode. You can specify to generate interrupts at the start of a (burst) conversion or at the end of a (burst) conversion when the conversion results are available to be read. Interrupts can also be generated if a failure occurs on the specified A/D converter.

The setting takes effect only after you have called `AdcCl1AnalogIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Tip

You can also specify trigger signals instead of interrupts, or a combination of both, by using `AdcCl1AnalogIn_setTriggerMode`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using `AdcCl1AnalogIn_create`.

InterruptMode Lets you specify the interrupt mode. The values can be combined by using the bitwise OR operator.

Symbol	Meaning
<code>ADC_CLASS1_NO_INTERRUPT</code>	Interrupts are not generated (default).
<code>ADC_CLASS1_INT_BURST_CONV_START</code>	Interrupts are generated on <ul style="list-style-type: none"> ▪ Burst start (when operating in burst conversion mode)

Symbol	Meaning
ADC_CLASS1_INT_BURST_CONV_END	<ul style="list-style-type: none"> ▪ Conversion start (when operating in single conversion mode) Interruptions are generated at the end of a ▪ Burst (when operating in burst conversion mode) ▪ Conversion (when operating in single conversion mode)
ADC_CLASS1_INT_CHANNEL_FAILURE	Interruptions are generated on channel failure.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_enableInterrupts.....	211
AdcCl1AnalogIn_getFailureInfo.....	216
AdcCl1AnalogIn_readFailureInfo.....	222
AdcCl1AnalogIn_setIoIntVector.....	234
AdcCl1AnalogIn_setTriggerMode.....	241

AdcCl1AnalogIn_setIoIntVector

Syntax

```
UInt32 AdcCl1AnalogIn_setIoIntVector(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 InterruptSelect
    IoLibTIntHandler InterruptHandler)
```

Include file

IoDrvAdcClass1AnalogIn.h

Purpose

To set the interrupt handler for processing interrupts from an ADC class 1 channel.

Description

This function is used to register the function that handles the generated interrupts specified by using **AdcCl1AnalogIn_setInterruptMode**. For each specified condition for generating an interrupt (conversion start, conversion end, failure), you can define separate functions. If you want to handle all the

conditions with the same function, you can use the bitwise OR operator to combine the predefined symbols of the `ConditionSelect` parameter.

The setting takes effect only after you have called `AdcCl1AnalogIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The `AdcCl1AnalogIn` driver object must already have been created by using `AdcCl1AnalogIn_create`.

InterruptSelect Lets you specify the Interrupts an interrupt handler is to be associated with. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
ADC_CLASS1_INT_BURST_CONV_START	The interrupt handler is to process interrupts generated on <ul style="list-style-type: none"> ▪ Conversion start (single conversion) ▪ Burst start (burst conversion)
ADC_CLASS1_INT_BURST_CONV_END	The interrupt handler is to process interrupts generated on <ul style="list-style-type: none"> ▪ Conversion end (single conversion) ▪ Burst end (burst conversion)
ADC_CLASS1_INT_CHANNEL_FAILURE	The interrupt handler is to process interrupts generated on a channel failure.

InterruptHandler Lets you specify the address of the function which is to handle the generated interrupt.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

AdcCl1AnalogIn_apply	206
AdcCl1AnalogIn_create	208
AdcCl1AnalogIn_enableInterrupts	211

AdcCl1AnalogIn_getFailureInfo.....	216
AdcCl1AnalogIn_readFailureInfo.....	222
AdcCl1AnalogIn_setTriggerMode.....	241

AdcCl1AnalogIn_setTimerPeriod

Syntax	<pre>UInt32 AdcCl1AnalogIn_setTimerPeriod(AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn, UInt32 TimerPeriod)</pre>
Include file	<code>IoDrvAdcClass1AnalogIn.h</code>
Purpose	To set the period of the timer of an ADC class 1 channel.
Description	<p>If you have specified the channel-specific timer to be used as trigger source for a conversion or a burst, you can use this function to change the default value during the initialization phase of the real-time application. The timer period specifies the time interval for generating conversion or burst triggers.</p> <p>The setting takes effect only after you have called <code>AdcCl1AnalogIn_apply</code> during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features).</p>
Parameters	<p>pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using <code>AdcCl1AnalogIn_create</code>.</p> <p>TimerPeriod Lets you specify the period of the channel-specific timer in seconds in the range 1 µs ... 1.34 s. You can specify the value in steps of 10 ns.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_setConversionMode.....	229
AdcCl1AnalogIn_write.....	249

AdcCl1AnalogIn_setTriggerLineIn

Syntax

```
UInt32 AdcCl1AnalogIn_setTriggerLineIn(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 TriggerLineIn)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To set the trigger line used as trigger source for an ADC class 1 channel.

Description

The A/D converter is able to *generate* trigger signals and to *consume* trigger signals as an optional source for burst and conversion triggers. With this function, you can specify the trigger line to which the A/D converter has to listen to get a trigger signal for starting conversions (single conversions and/or bursts of conversions) specified by using `AdcCl1AnalogIn_setConversTrigger` and/or `AdcCl1AnalogIn_setBurstTrigger`.

The setting takes effect only after you have called `AdcCl1AnalogIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using [AdcCl1AnalogIn_create](#).

TriggerLineIn Lets you specify the number of the trigger line used as trigger source in the range 1 ... 16.

Symbol	Meaning
ADC_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.
...	...
ADC_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_setBurstTrigger.....	226
AdcCl1AnalogIn_setConversTrigger.....	231
AdcCl1AnalogIn_setTriggerLineOut.....	238

AdcCl1AnalogIn_setTriggerLineOut

Syntax

```
UInt32 AdcCl1AnalogIn_setTriggerLineOut(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 TriggerLineOut)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To set the trigger line used for transmitting a generated trigger signal.

Description	<p>A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board (internal trigger signal) or to a digital output channel (external trigger signal). You have to specify which trigger line the consumer must listen to by using the relevant <code>xxx_SetTriggerLineIn</code> function. A trigger line can be allocated only once, but multiple consumers can listen to it.</p> <p>Before you can use trigger signals for A/D conversion, you have to set the trigger mode by using <code>AdcCl1AnalogIn_SetTriggerMode</code>. With <code>AdcCl1AnalogIn_SetTriggerLineOutStatus</code> you enable or disable the trigger line.</p> <p>The setting takes effect only after you have called <code>AdcCl1AnalogIn_Apply</code> during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>								
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features).</p>								
Parameters	<p>pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The <code>AdcCl1AnalogIn</code> driver object must already have been created by using <code>AdcCl1AnalogIn_Create</code>.</p> <p>TriggerLineOut Lets you specify the number of the trigger line used for transmitting a generated trigger signal in the range 1 ... 16.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>ADC_CLASS1_TRIGGER_LINE_1</td><td>Specifies trigger line 1.</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td>ADC_CLASS1_TRIGGER_LINE_16</td><td>Specifies trigger line 16.</td></tr> </tbody> </table>	Symbol	Meaning	ADC_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.	ADC_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.
Symbol	Meaning								
ADC_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.								
...	...								
ADC_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.								
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.		
Error Code	Meaning								
IOLIB_NO_ERROR	The function was successfully completed.								
!= IOLIB_NO_ERROR	The function was not completed.								
Related topics	<p>References</p> <table border="1"> <tr> <td>AdcCl1AnalogIn_Apply.....</td><td>206</td></tr> <tr> <td>AdcCl1AnalogIn_Create.....</td><td>208</td></tr> </table>	AdcCl1AnalogIn_Apply	206	AdcCl1AnalogIn_Create	208				
AdcCl1AnalogIn_Apply	206								
AdcCl1AnalogIn_Create	208								

AdcCl1AnalogIn_setTriggerLineIn.....	237
AdcCl1AnalogIn_setTriggerLineOutStatus.....	240
AdcCl1AnalogIn_setTriggerMode.....	241

AdcCl1AnalogIn_setTriggerLineOutStatus

Syntax

```
UInt32 AdcCl1AnalogIn_setTriggerLineOutStatus(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 TriggerLineStatus)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To set the status of the trigger line.

Description

With this function, you can enable or disable the trigger line that you specified before by using `AdcCl1AnalogIn_setTriggerLineOut`.

The function is intended to be called during the initialization phase of the real-time application or during a run-to-stop or a stop-to-run transition of the real-time application.

The setting does not take effect until you call `AdcCl1AnalogIn_start` or `AdcCl1AnalogIn_stop`.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using `AdcCl1AnalogIn_create`.

TriggerLineStatus Lets you specify the status of the trigger line.

Symbol	Meaning
ADC_CLASS1_TRIGGER_LINE_DISABLE	Disables the trigger line.
ADC_CLASS1_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_start.....	243
AdcCl1AnalogIn_stop.....	244

AdcCl1AnalogIn_setTriggerMode

Syntax

```
UInt32 AdcCl1AnalogIn_setTriggerMode(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn,
    UInt32 TriggerMode)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To set the trigger mode of the ADC class 1 channel.

Description

The trigger mode is used for single conversion mode and burst conversion mode. You can specify to generate a trigger signal at the start of a (burst) conversion or at the end of a (burst) conversion when the conversion results are available to be read. A trigger signal can also be generated if a failure occurred on the specified A/D converter.

Trigger signals always require a generator for the signal and a consumer that listens to the trigger line. These components are specified by using the `xxx_setTriggerLineOut` function, for example, `AdcCl1AnalogIn_setTriggerLineOut`, and the `xxx_setTriggerLineIn` function.

The setting takes effect only after you have called `AdcCl1AnalogIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Tip

As an alternative, you can specify interrupts instead of trigger signals, or a combination of both, by using `AdcCl1AnalogIn_setInterruptMode`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using `AdcCl1AnalogIn_create`.

TriggerMode Lets you specify the trigger mode.

Symbol	Meaning
ADC_CLASS1_NO_TRIGGER	Trigger signals are not generated (default).
ADC_CLASS1_TRIG_BURST_CONV_START	Trigger signals are generated on <ul style="list-style-type: none"> ▪ Burst start (when operating in burst conversion mode) ▪ Conversion start (when operating in single conversion mode)
ADC_CLASS1_TRIG_BURST_CONV_END	Trigger signals are generated at the end of a <ul style="list-style-type: none"> ▪ Burst (when operating in burst conversion mode) ▪ Conversion (when operating in single conversion mode)
ADC_CLASS1_TRIG_CHANNEL_FAILURE	Trigger signals are generated on channel failure.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_setInterruptMode.....	233
AdcCl1AnalogIn_setTriggerLineIn.....	237
AdcCl1AnalogIn_setTriggerLineOut.....	238

AdcCl1AnalogIn_start

Syntax

```
UInt32 AdcCl1AnalogIn_start(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To enable the driver object to react on a conversion trigger.

Description

This function starts the specified AdcCl1AnalogIn driver so that it will activate A/D conversion. The analog input channel is enabled to sample the input signals according to the specified trigger sources. Afterwards, the conversion results can be read. This function also enables the configured trigger lines. If you call `AdcCl1AnalogIn_setTriggerLineOutStatus(ADC_CLASS1_TRIGGER_LINE_DISABLE)` before, the trigger line stays disabled.

The function is intended to be called during run time of the real-time application.

Before you call `AdcCl1AnalogIn_start` again in your application, you have to call `AdcCl1AnalogIn_stop` to disable the driver.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using `AdcCl1AnalogIn_create`.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics

References

AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_setTriggerLineOutStatus.....	240
AdcCl1AnalogIn_stop.....	244

[AdcCl1AnalogIn_stop](#)

Syntax

```
UInt32 AdcCl1AnalogIn_stop(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn)
```

Include file

IoDrvAdcClass1AnalogIn.h

Purpose

To disable the driver object to react on a conversion trigger.

Description

This function stops the specified AdcCl1AnalogIn driver so that it will deactivate A/D conversion. The analog input channel is disabled to sample the input signals according to the specified trigger sources and the trigger line stays enabled. If you call

`AdcCl1AnalogIn_setTriggerLineOutStatus(ADC_CLASS1_TRIGGER_LINE_DISABLE)` before, `AdcCl1AnalogIn_stop` disables the trigger line.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using `AdcCl1AnalogIn_create`.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_setTriggerLineOutStatus.....	240
AdcCl1AnalogIn_start.....	243

AdcCl1AnalogIn_triggerBurst

Syntax

```
UInt32 AdcCl1AnalogIn_triggerBurst(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To generate a burst trigger for one ADC class 1 channel.

Description

If you have specified to operate in burst conversion mode and the burst start is configured to be triggered by software, this function generates the trigger signal. The associated channel is specified by its I/O driver.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#) .

Parameters

pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using [AdcCl1AnalogIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_triggerBurstMultiple.....	246
AdcCl1AnalogIn_triggerConversion.....	247
AdcCl1AnalogIn_write.....	249

AdcCl1AnalogIn_triggerBurstMultiple

Syntax

```
UInt32 AdcCl1AnalogIn_triggerBurstMultiple(UInt32 ChannelMask)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To generate a burst trigger for multiple ADC class 1 channels.

Description

If you have specified to operate in burst conversion mode, and the burst start is configured to be triggered by software, you can use this function to synchronously generate the trigger signal for several channels. The associated channels must be specified in a channel mask.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

ChannelMask Lets you specify the channels to be triggered in the range 1 ... 16. You can combine the predefined symbols by using the bitwise OR operator.

Symbol	Meaning
ADC_CLASS1_MASK_CH_1	Specifies channel 1.
...	...
ADC_CLASS1_MASK_CH_16	Specifies channel 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_write.....	249

AdcCl1AnalogIn_triggerConversion

Syntax

```
UInt32 AdcCl1AnalogIn_triggerConversion(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn)
```

Include file

IoDrvAdcClass1AnalogIn.h

Purpose

To generate a conversion trigger for one ADC class 1 channel.

Description

If you have specified to start a conversion by software, you can use this function to trigger a conversion for one ADC class 1 channel. The associated channel is specified by its I/O driver.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration ).
	For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features)  .
Parameters	pAdcCl1AnalogIn Lets you specify the ADC Class 1 channel to be used. The AdcCl1AnalogIn driver object must already have been created by using AdcCl1AnalogIn_create .
Return value	The function returns an error code.

Related topics	References
	AdcCl1AnalogIn_apply 206
	AdcCl1AnalogIn_create 208
	AdcCl1AnalogIn_write 249

AdcCl1AnalogIn_triggerConversionMultiple

Syntax	<code>UInt32 AdcCl1AnalogIn_triggerConversionMultiple(UInt32 ChannelMask)</code>
Include file	<code>IoDrvAdcClass1AnalogIn.h</code>
Purpose	To generate a conversion trigger for multiple ADC class 1 channels.
Description	If you have specified to start a conversion by software, you can use this function to synchronously start a conversion on several A/D channels. The associated channels must be specified in a channel mask. If you want to trigger a single channel, it is recommended to use AdcCl1AnalogIn_triggerConversion . The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#)
[\(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 1 \(MicroLabBox Features\)](#).

Parameters

ChannelMask Lets you specify the channels to be triggered in the range 1 ... 16. You can combine the predefined symbols by using the bitwise OR operator.

Symbol	Meaning
ADC_CLASS1_MASK_CH_1	Specifies channel 1.
...	...
ADC_CLASS1_MASK_CH_16	Specifies channel 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

AdcCl1AnalogIn_apply.....	206
AdcCl1AnalogIn_create.....	208
AdcCl1AnalogIn_write.....	249

AdcCl1AnalogIn_write

Syntax

```
UInt32 AdcCl1AnalogIn_write(
    AdcCl1AnalogInSDrvObject *pAdcCl1AnalogIn)
```

Include file

`IoDrvAdcClass1AnalogIn.h`

Purpose

To update settings of the specified ADC class 1 channel during run time.

Description	If you used <code>AdcCl1AnalogIn_setChannelState</code> during run time, its settings will not be updated until you have called <code>AdcCl1AnalogIn_write</code> . The specified analog input channel must be enabled by using <code>AdcCl1AnalogIn_start</code> . If an error is detected, the very first instance of the error is returned to the caller.
I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) . For more information on the I/O mapping, refer to ADC Class 1 (MicroLabBox Features) .
Parameters	<code>pAdcCl1AnalogIn</code> Lets you specify the ADC Class 1 channel to be used. The <code>AdcCl1AnalogIn</code> driver object must already have been created by using <code>AdcCl1AnalogIn_create</code> .
Return value	The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics	References
	<p>AdcCl1AnalogIn_create..... 208</p> <p>AdcCl1AnalogIn_start..... 243</p>

Analog In Class 2

Introduction	The Analog In Class 2 unit can be accessed via the <code>AdcCl2AnalogIn</code> functions.
Where to go from here	Information in this section

AdcCl2AnalogIn_apply	251
To apply the initialization settings to the ADC class 2 channel.	

AdcCl2AnalogIn_create	252
To create the I/O driver object for analog signal inputs via the ADC Class 2 unit.	
AdcCl2AnalogIn_getInputValue	254
To get the measured value from the ADC Class 2 channel.	
AdcCl2AnalogIn_read	255
To read the analog measurement value from the ADC class 2 channel.	
AdcCl2AnalogIn_start	256
To enable the driver to start A/D conversion on the specified ADC Class 2 channel.	
Example of Implementing A/D Conversion Using ADC Class 2	257
Shows you how to implement A/D conversion using the class 2 analog input channels.	

Information in other sections

[ADC Class 2 \(MicroLabBox Features\)](#) 

AdcCl2AnalogIn_apply

Syntax

```
UInt32 AdcCl2AnalogIn_apply(
    AdcCl2AnalogInSDrvObject *pAdcCl2AnalogIn)
```

Include file

`IoDrvAdcClass2AnalogIn.h`

Purpose

To apply the initialization settings to the ADC class 2 channel.

Description

Before you can start the specified A/D converter, you have to transfer the settings of the `AdcCl2AnalogIn` driver object to its related input channels by using `AdcCl2AnalogIn_apply`.

The `AdcCl2AnalogIn_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The signal measurement on the specified analog input channel is not activated until you have called `AdcCl2AnalogIn_start`.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration ).						
	For more information on the I/O mapping, refer to ADC Class 2 (MicroLabBox Features ).						
Parameters	pAdcCl2AnalogIn Lets you specify the ADC Class 2 channel to be used. The AdcCl2AnalogIn driver object must already have been created by using AdcCl2AnalogIn_create .						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	<p>Examples</p> <div style="background-color: #f0f0f0; padding: 5px;"> Example of Implementing A/D Conversion Using ADC Class 2..... 257 </div> <p>References</p> <div style="background-color: #f0f0f0; padding: 5px;"> AdcCl2AnalogIn_create..... 252 AdcCl2AnalogIn_start..... 256 </div>						

AdcCl2AnalogIn_create

Syntax	<pre>UInt32 AdcCl2AnalogIn_create(AdcCl2AnalogInSDrvObject **ppAdcCl2AnalogIn, UInt32 Channel)</pre>
Include file	<code>IoDrvAdcClass2AnalogIn.h</code>
Purpose	To create the I/O driver object for analog signal inputs via the ADC Class 2 unit.
Description	This function performs all the steps necessary to create an I/O driver object for the ADC Class 2 unit. It specifies one analog input channel for signal

measurement. The instantiated ADC Class 2 converter operates in free-running mode and provides conversion results continuously.

The ADC Class 2 converter is initialized with default values so that it is ready for use.

The function is intended to be called during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 2 \(MicroLabBox Features\)](#).

Parameters

ppAdcCl2AnalogIn Lets you specify the address of a variable which is to hold the address of the created driver object.

Channel Lets you specify the channel to be accessed by this driver in the range 1 ... 8.

Symbol	Meaning
ADC_CLASS2_CHANNEL_1	Specifies channel 1 of the ADC Class 2 unit.
...	...
ADC_CLASS2_CHANNEL_8	Specifies channel 8 of the ADC Class 2 unit.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

Examples

[Example of Implementing A/D Conversion Using ADC Class 2](#)..... 257

References

[AdcCl2AnalogIn_apply](#)..... 251

AdcCl2AnalogIn_getInputValue

Syntax

```
UInt32 AdcCl2AnalogIn_getInputValue(
    AdcCl2AnalogInSDrvObject *pAdcCl2AnalogIn,
    Float64 *pInputValue)
```

Include file

`IoDrvAdcClass2AnalogIn.h`

Purpose

To get the measured value from the ADC Class 2 channel.

Description

Because you get the value from the internal buffer, there might be a newer value if a conversion is in progress. To update the internal buffer with the latest value, you first have to call `AdcCl2AnalogIn_read`. However, as long as the current conversion is not completed, the read value comes from the preceding conversion.

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

Note

To get a conversion result, you must enable the specified ADC Class 2 channel via `AdcCl2AnalogIn_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [ADC Class 2 \(MicroLabBox Features\)](#) .

Parameters

pAdcCl2AnalogIn Lets you specify the ADC Class 2 channel to be used. The `AdcCl2AnalogIn` driver object must already have been created by using `AdcCl2AnalogIn_create`.

pInputValue Lets you specify the address of a variable that holds the measured value in the range -1.0 ... +1.0 with a resolution of 14 bits.

Value	Voltage
-1.0	-10.0 V
...	...
0.0	0.0 V

Value	Voltage
...	...
+1.0	+10 V

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**Examples**

[Example of Implementing A/D Conversion Using ADC Class 2](#)..... 257

References

AdcCl2AnalogIn_apply.....	251
AdcCl2AnalogIn_create.....	252
AdcCl2AnalogIn_read.....	255
AdcCl2AnalogIn_start.....	256

AdcCl2AnalogIn_read

Syntax

```
UInt32 AdcCl2AnalogIn_read(AdcCl2AnalogInSDrvObject *pAdcCl2AnalogIn)
```

Include file

`IoDrvAdcClass2AnalogIn.h`

Purpose

To read the analog measurement value from the ADC class 2 channel.

Description

This function reads the raw data from the specified analog input channel and calculates the corresponding conversion result, which is then stored in the internal buffer of the driver object. You can get the value from the internal buffer by using [AdcCl2AnalogIn_getInputValue](#).

To use this function, you must enable the analog input channel via [AdcCl2AnalogIn_start](#).

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#)
[\(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [ADC Class 2 \(MicroLabBox Features\)](#).

Parameters

pAdcCl2AnalogIn Lets you specify the ADC Class 2 channel to be used. The AdcCl2AnalogIn driver object must already have been created by using [AdcCl2AnalogIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**Examples**

[Example of Implementing A/D Conversion Using ADC Class 2](#)..... 257

References

AdcCl2AnalogIn_apply.....	251
AdcCl2AnalogIn_create.....	252
AdcCl2AnalogIn_getInputValue.....	254
AdcCl2AnalogIn_start.....	256

AdcCl2AnalogIn_start

Syntax

```
UInt32 AdcCl2AnalogIn_start(
    AdcCl2AnalogInSDrvObject *pAdcCl2AnalogIn)
```

Include file

[IoDrvAdcClass2AnalogIn.h](#)

Purpose

To enable the driver to start A/D conversion on the specified ADC Class 2 channel.

Description	This function starts the specified AdcCl2AnalogIn driver so that it will activate A/D conversion. Then you can read the current conversion result by using AdcCl2AnalogIn_read . The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.						
I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) . For more information on the I/O mapping, refer to ADC Class 2 (MicroLabBox Features) .						
Parameters	pAdcCl2AnalogIn Lets you specify the ADC Class 2 channel to be used. The AdcCl2AnalogIn driver object must already have been created by using AdcCl2AnalogIn_create .						
Return value	The function returns an error code.						
Related topics	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Examples</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">Example of Implementing A/D Conversion Using ADC Class 2..... 257</td></tr> </tbody> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">References</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">AdcCl2AnalogIn_apply..... 251</td></tr> <tr> <td style="padding: 2px;">AdcCl2AnalogIn_create..... 252</td></tr> <tr> <td style="padding: 2px;">AdcCl2AnalogIn_read..... 255</td></tr> </tbody> </table>	Examples	Example of Implementing A/D Conversion Using ADC Class 2 257	References	AdcCl2AnalogIn_apply 251	AdcCl2AnalogIn_create 252	AdcCl2AnalogIn_read 255
Examples							
Example of Implementing A/D Conversion Using ADC Class 2 257							
References							
AdcCl2AnalogIn_apply 251							
AdcCl2AnalogIn_create 252							
AdcCl2AnalogIn_read 255							

Example of Implementing A/D Conversion Using ADC Class 2

Introduction	The following example code shows you the order in which the functions are to be used. The code cannot be built without modifications.
---------------------	---

Using ADC Class 2 channels

The example code shows you how to use channel 1 of the ADC class 2 unit for A/D conversion.

```
AdcCl2AnalogInSDrvObject* MyADCCL2;
UIInt32 IErr;
Float64 Data;
...
/* Initialize the A/D converter */
/* Create the driver object that is using channel 1 of the ADC class 2 unit. */
IErr=AdcCl2AnalogIn_create(&MyADCCL2, ADC_CLASS2_CHANNEL_1);
/* Apply the default settings to the driver object. */
IErr=AdcCl2AnalogIn_apply(MyADCCL2);
/* Enable the A/D channel */
IErr=AdcCl2AnalogIn_start(MyADCCL2);
/* Read the current conversion result from the channel */
while()
{
    IErr=AdcCl2AnalogIn_read(MyADCCL2);
    /* Get the measured value from the internal buffer. */
    IErr=AdcCl2AnalogIn_getInputValue(MyADCCL2, &Data);
...
}
```

D/A Conversion

Introduction

MicroLabBox provides one I/O unit for D/A conversion.

For basic information on MicroLabBox's DAC unit, refer to [D/A Conversion](#) ([MicroLabBox Features](#)).

Where to go from here

Information in this section

[Analog Out Class 1](#)..... 259

Information in other sections

[Implementing I/O Functions](#)..... 198

Analog Out Class 1

Introduction

The Analog Out Class 1 unit can be accessed via the [AdcCl1AnalogOut](#) functions.

Where to go from here

Information in this section

[DacCl1AnalogOut_apply](#)..... 260

To apply the initialization settings to the specified DAC Class 1 channel(s).

[DacCl1AnalogOut_create](#)..... 261

To create the I/O driver object for analog signal outputs via DAC Class 1 unit.

[DacCl1AnalogOut_setOutputValue](#)..... 263

To set the value to be output by one channel of the DAC Class 1 unit.

[DacCl1AnalogOut_start](#)..... 264

To activate D/A conversion on the specified DAC Class 1 channel(s).

[DacCl1AnalogOut_write](#)..... 266

To write the specified values to the DAC Class 1 channel(s).

[Example of Implementing D/A Conversion Using DAC Class 1](#) 267
Shows you how to implement D/A conversion using the DAC class 1 unit.

Information in other sections

[DAC Class 1 \(MicroLabBox Features\)](#)

DacCl1AnalogOut_apply

Syntax

```
UInt32 DacCl1AnalogOut_apply(  
    DacCl1AnalogOutSDrvObject *pDacCl1AnalogOut)
```

Include file

`IoDrvDacClass1AnalogOut.h`

Purpose

To apply the initialization settings to the specified DAC Class 1 channel(s).

Description

Before you can start the specified D/A converter, you have to transfer the settings of the DacCl1AnalogOut driver object to its related output channel(s) by using **DacCl1AnalogOut_apply**. You have to do this also for the default values and for values that you specified by using the **DacCl1AnalogOut_set<Parameter>** functions. The **DacCl1AnalogOut_apply** function is intended to be called at the end of the initialization phase of the real-time application.

The signal generation on the specified analog output channel(s) is not activated until you have called **DacCl1AnalogOut_start**.

To update settings during run time, you have to use **DacCl1AnalogOut_write**.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [DAC Class 1 \(MicroLabBox Features\)](#).

Parameters	pDacCl1AnalogOut Lets you specify the D/A channel(s) to be used. The DacCl1AnalogOut driver object must already have been created by using DacCl1AnalogOut_create .
-------------------	---

Return value	The function returns an error code.
---------------------	-------------------------------------

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	Examples
	Example of Implementing D/A Conversion Using DAC Class 1..... 267
	References
	DacCl1AnalogOut_create..... 261 DacCl1AnalogOut_start..... 264 DacCl1AnalogOut_write..... 266

DacCl1AnalogOut_create

Syntax	<code>UInt32 DacCl1AnalogOut_create(DacCl1AnalogOutSDrvObject **ppDacCl1AnalogOut, UInt32 ChannelMask)</code>
---------------	--

Include file	<code>IoDrvDacClass1AnalogOut.h</code>
---------------------	--

Purpose	To create the I/O driver object for analog signal outputs via DAC Class 1 unit.
----------------	---

Description	<p>The DacCl1AnalogOut driver object can be used to handle up to 16 analog output channels which can be specified via a channel mask.</p> <p>The DAC Class 1 converter is initialized with default values so that it is ready for use.</p> <p>The standard values are as follows:</p> <ul style="list-style-type: none"> ▪ Output value: 0.0
--------------------	---

The function is intended to be called during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [DAC Class 1 \(MicroLabBox Features\)](#).

Parameters

ppDacCl1AnalogOut Lets you specify the address of a variable which is to hold the address of the created driver object.

ChannelMask Lets you select the channel(s) to be accessed by this driver in the range 1 ... 16. To select more than one channel, combine the predefined symbols by using the bitwise OR operator.

Symbol	Meaning
DAC_CLASS1_MASK_CH_1	Specifies channel 1 of the DAC Class 1 unit.
...	...
DAC_CLASS1_MASK_CH_16	Specifies channel 16 of the DAC Class 1 unit.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**Examples**

[Example of Implementing D/A Conversion Using DAC Class 1](#)..... 267

References

[DacCl1AnalogOut_apply](#)..... 260

DacCl1AnalogOut_setOutputValue

Syntax

```
UInt32 DacCl1AnalogOut_setOutputValue(
    DacCl1AnalogOutSDrvObject *pDacCl1AnalogOut,
    UInt32 Channel
    Float64 OutputValue)
```

Include file

`IoDrvDacClass1AnalogOut.h`

Purpose

To set the value to be output by one channel of the DAC Class 1 unit.

Description

This function is used to specify the value that you want to output on the specified D/A channel. The value is written to the internal buffer of the driver object.

The setting does not take effect until you call `DacCl1AnalogOut_apply` during the initialization phase or `DacCl1AnalogOut_write` during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Note

All analog output channels of the DAC Class 1 unit are in a disabled state after reset. To enable the channel(s) controlled by this driver, use the `DacCl1AnalogOut_start` function.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [DAC Class 1 \(MicroLabBox Features\)](#).

Parameters

pDacCl1AnalogOut Lets you specify the D/A channel(s) to be used. The DacCl1AnalogOut driver object must already have been created by using `DacCl1AnalogOut_create`.

Channel Lets you specify the channel on which the value is to be output in the range 1 ... 16. The specified channel must be contained in the channel mask you used when creating the driver object.

Symbol	Meaning
DAC_CLASS1_CHANNEL_1	Specifies channel 1 of the DAC Class 1 unit.
...	...
DAC_CLASS1_CHANNEL_16	Specifies channel 16 of the DAC Class 1 unit.

OutputValue Lets you specify the value to be generated on the analog output channel in the range -1.0 ... +1.0. The resolution of the value is 16 bits. The following table shows the mapping between the specified value and the corresponding output voltage in V.

Value	Voltage
-1.0	-10.0 V
...	...
0.0	0.0 V
...	...
+1.0	+10 V

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**Examples**

[Example of Implementing D/A Conversion Using DAC Class 1](#)..... 267

References

DacCl1AnalogOut_apply	260
DacCl1AnalogOut_create	261
DacCl1AnalogOut_write	266

[DacCl1AnalogOut_start](#)**Syntax**

```
UInt32 DacCl1AnalogOut_start(DacCl1AnalogOutSDrvObject  
*pDacCl1AnalogOut)
```

Include file

`IoDrvDacClass1AnalogOut.h`

Purpose	To activate D/A conversion on the specified DAC Class 1 channel(s).						
Description	<p>This function starts the specified DacCl1AnalogOut driver so that it will activate D/A conversion. The analog output channels are enabled to generate the output signals according to the specified values.</p> <p>The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.</p>						
	<div style="background-color: #f0f0f0; padding: 10px;"> <p>Note</p> <p>Because all analog output channels of the DAC Class 1 unit are in a disabled state after reset (output voltage is set to 0.0 V), this function must be called to enable the output channels and drive the outputs according to the requested analog output signal.</p> </div>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to DAC Class 1 (MicroLabBox Features).</p>						
Parameters	<p>pDacCl1AnalogOut Lets you specify the D/A channel(s) to be used. The DacCl1AnalogOut driver object must already have been created by using DacCl1AnalogOut_create.</p>						
Return value	<p>The function returns an error code.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Error Code</th> <th style="text-align: left; padding: 2px;">Meaning</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">IOLIB_NO_ERROR</td> <td style="padding: 2px;">The function was successfully completed.</td> </tr> <tr> <td style="padding: 2px;">!= IOLIB_NO_ERROR</td> <td style="padding: 2px;">The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	<p>Examples</p> <div style="background-color: #f0f0f0; padding: 10px;"> <p>Example of Implementing D/A Conversion Using DAC Class 1..... 267</p> </div> <p>References</p> <div style="background-color: #f0f0f0; padding: 10px;"> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">DacCl1AnalogOut_apply.....</td> <td style="width: 10%;">260</td> </tr> <tr> <td>DacCl1AnalogOut_create.....</td> <td>261</td> </tr> <tr> <td>DacCl1AnalogOut_setOutputValue.....</td> <td>263</td> </tr> </table> </div>	DacCl1AnalogOut_apply.....	260	DacCl1AnalogOut_create.....	261	DacCl1AnalogOut_setOutputValue.....	263
DacCl1AnalogOut_apply.....	260						
DacCl1AnalogOut_create.....	261						
DacCl1AnalogOut_setOutputValue.....	263						

DacCl1AnalogOut_write

Syntax

```
UInt32 DacCl1AnalogOut_write(
    DacCl1AnalogOutSDrvObject *pDacCl1AnalogOut)
```

Include file

`IoDrvDacClass1AnalogOut.h`

Purpose

To write the specified values to the DAC Class 1 channel(s).

Description

The values that you specified for the DAC Class 1 channels by using `DacCl1AnalogOut_setOutputValue`, are output simultaneously.

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Note

- The outputs are generated only on those channels that you specified in the channel mask of the driver object.
- All analog output channels of the DAC Class 1 unit are in a disabled state after reset. To enable the channel(s) controlled by this driver, use the `DacCl1AnalogOut_start` function.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [DAC Class 1 \(MicroLabBox Features\)](#).

Parameters

`pDacCl1AnalogOut` Lets you specify the D/A channel(s) to be used. The DacCl1AnalogOut driver object must already have been created by using `DacCl1AnalogOut_create`.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics**Examples**

[Example of Implementing D/A Conversion Using DAC Class 1](#)..... 267

References

DacCl1AnalogOut_create.....	261
DacCl1AnalogOut_setOutputValue.....	263
DacCl1AnalogOut_start.....	264

Example of Implementing D/A Conversion Using DAC Class 1

Introduction

The following example code shows you the order in which the functions are to be used. The code cannot be built without modifications.

Using DAC Class 1 channels

The example code shows you how to use channels 1 and 2 of the DAC class 1 unit for D/A conversion.

```

DacCl1AnalogOutSDrvObject* MyDACCL1;
UInt32 IErr;

...
/* Initialize the D/A converter */
/* Create the driver object that is using channels 1 and 2 of the DAC class 1 unit. */
IErr=DacCl1AnalogOut_create(&MyDACCL1, DAC_CLASS1_MASK_CH_1|DAC_CLASS1_MASK_CH_2);
/* Set the output value for channel 1 to 5 V (=0.5) */
IErr=DacCl1AnalogOut_setOutputValue(MyDACCL1, DAC_CLASS1_CHANNEL_1, 0.5);
/* Set the output value for channel 2 to 5 V (=0.5) */
IErr=DacCl1AnalogOut_setOutputValue(MyDACCL1, DAC_CLASS1_CHANNEL_2, 0.5);
/* Apply the default settings to the driver object. */
IErr=DacCl1AnalogOut_apply(MyDACCL1);
/* Start the driver */
IErr=DacCl1AnalogOut_start(MyDACCL1);
...

```

Related topics**References**

DacCl1AnalogOut_apply.....	260
DacCl1AnalogOut_create.....	261
DacCl1AnalogOut_setOutputValue.....	263
DacCl1AnalogOut_start.....	264

Bit I/O

Introduction	MicroLabBox provides functions for accessing the digital I/O channels of Class 1 for bit I/O. For basic information on MicroLabBox's Bit I/O Class 1 unit, refer to Bit I/O (MicroLabBox Features ).
---------------------	--

Where to go from here	Information in this section
	Digital In Class 1..... 268
	Digital Out Class 1..... 290
	Information in other sections
	Implementing I/O Functions..... 198

Digital In Class 1

Introduction	The Digital In Class 1 unit for bit I/O can be accessed via the DioCl1DigIn functions.
---------------------	--

Where to go from here	Information in this section
	DioCl1DigIn_apply..... 269 To apply the initialization settings to the DIO Class 1 input channels.
	DioCl1DigIn_create..... 270 To create the I/O driver object for digital signal inputs via the DIO Class 1 unit.
	DioCl1DigIn_disableInterrupts..... 272 To disable the generation of interrupts on a DIO Class 1 input channel.
	DioCl1DigIn_enableInterrupts..... 273 To enable the generation of interrupts on a DIO Class 1 input channel.
	DioCl1DigIn_getChMaskInData..... 274 To get the values read from digital input channels.

DioCl1DigIn_getPortInData	275
To get the values read from a digital input port.	
DioCl1DigIn_read	276
To read data from the DIO Class 1 input channels.	
DioCl1DigIn_setEventDelay	278
To set the delay for events on a DIO Class 1 input channel.	
DioCl1DigIn_setEventDownSample	279
To set the downsampling for events on a DIO Class 1 input channel.	
DioCl1DigIn_setInputFilter	280
To set the input filter of the DIO Class 1 unit.	
DioCl1DigIn_setInterruptMode	281
To set the interrupt mode of a DIO Class 1 input channel.	
DioCl1DigIn_setIntVect	282
To set the interrupt handler for processing interrupts from a DIO Class 1 input channel.	
DioCl1DigIn_setTriggerLineOut	284
To select the trigger line used for trigger signal generation.	
DioCl1DigIn_setTriggerLineOutStatus	285
To set the status of the trigger line.	
DioCl1DigIn_setTriggerMode	286
To set the trigger mode of a DIO Class 1 input channel.	
DioCl1DigIn_start	288
To start the DIO Class 1 input channels.	
Example of Implementing Bit I/O Inputs Using DIO Class 1	289
Shows you how to implement access to the digital input channels of the DIO Class 1 unit.	

Information in other sections

[Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#)

DioCl1DigIn_apply

Syntax

```
UInt32 DioCl1DigIn_apply(DioCl1DigInSDrvObject *pDioCl1DigIn)
```

Include file

IoDrvDioClass1BitIn.h

Purpose	To apply the initialization settings to the DIO Class 1 input channels.						
Description	<p>Before you can start the specified DIO Class 1 unit, you have to transfer the settings of the DioCl1DigIn driver object to its related input channels by using <code>DioCl1DigIn_apply</code>. You have to do this also for the default values and for values that you specified by using the <code>DioCl1DigIn_set<Parameter></code> functions. The <code>DioCl1DigIn_apply</code> function is intended to be called at the end of the initialization phase of the real-time application.</p> <p>The signal measurement on the specified digital input channel(s) is not activated until you have called <code>DioCl1DigIn_start</code>.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Bit I/O (DIO Class 1 (MicroLabBox Features)).</p>						
Parameters	<p><code>pDioCl1DigIn</code> Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using <code>DioCl1DigIn_create</code>.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						

DioCl1DigIn_create

Syntax	<pre>UInt32 DioCl1DigIn_create(DioCl1DigInSDrvObject **ppDioCl1DigIn, UInt32 Port, UInt32 ChannelMask)</pre>
Include file	<code>IoDrvDioClass1BitIn.h</code>

Purpose To create the I/O driver object for digital signal inputs via the DIO Class 1 unit.

Description This function performs all the steps necessary to create a bit I/O driver object for the DIO Class 1 unit. The 48 digital channels of the board are separated into three ports with 16 channels each. A DioCl1DigIn driver object can only handle the channels of one port. You can therefore specify up to 16 channels to be used with the current driver object via a channel mask.

The specified DIO Class 1 channels are initialized with default values so that they are ready for use.

The initial values are:

- Input filter: 0 s
- Interrupt mode: no interrupt generation
- Trigger mode: no trigger signal generation
- Event downsample: 1
- Event delay: 0 s

The function is intended to be called during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\)](#) ([MicroLabBox Features](#) ).

Parameters **ppDioCl1DigIn** Lets you specify the address of a variable which holds the address of the created driver object.

Port Lets you specify the number of the DIO Class 1 port providing the required channels in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
DIO_CLASS1_PORT_2	Specifies port 2 of the DIO Class 1 unit.
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

ChannelMask Lets you specify the channel(s) of the specified port to be controlled by the driver in the range 1 ... 16. To specify more than one channel, you can combine the symbols by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_MASK_CH_1	Specifies channel 1 of the DIO Class 1 unit.
...	...
DIO_CLASS1_MASK_CH_16	Specifies channel 16 of the DIO Class 1 unit.

Tip

To specify all the 16 channels of a port, you can enter `0xFFFF` as the channel mask.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

DioCl1DigIn_disableInterrupts

Syntax

```
UInt32 DioCl1DigIn_disableInterrupts(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    UInt32 InterruptSelect)
```

Include file

`IoDrvDioClass1BitIn.h`

Purpose

To disable the generation of interrupts on a DIO Class 1 input channel.

Description

With this function, you can disable the generation of interrupts that you specified before by using `DioCl1DigIn_setInterruptMode`. The settings take effect immediately.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using [DioCl1DigIn_create](#).

InterruptSelect Lets you select the interrupts that are to be disabled. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_RISING_EDGE	Disables the generation of interrupts on rising edges.
DIO_CLASS1_INT_FALLING_EDGE	Disables the generation of interrupts on falling edges.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigIn_enableInterrupts

Syntax

```
UInt32 DioCl1DigIn_enableInterrupts(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    UInt32 InterruptSelect)
```

Include file

`IoDrvDioClass1BitIn.h`

Purpose

To enable the generation of interrupts on a DIO Class 1 input channel.

Description

With this function, you can either enable the generation of interrupts or reenable the generation of interrupts that you disabled before by using [DioCl1DigIn_disableInterrupts](#). The interrupts have to be specified before by using [DioCl1DigIn_setInterruptMode](#). The settings take effect immediately.

After calling [DioCl1DigIn_start](#), interrupts are disabled.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using [DioCl1DigIn_create](#).

InterruptSelect Lets you select the interrupts that are to be enabled. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_RISING_EDGE	Enables the generation of interrupts on rising edges.
DIO_CLASS1_INT_FALLING_EDGE	Enables the generation of interrupts on falling edges.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1DigIn_disableInterrupts.....	272
DioCl1DigIn_setInterruptMode.....	281

DioCl1DigIn_getChMaskInData

Syntax

```
UInt32 DioCl1DigIn_getChMaskInData(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    UInt32 *pInputData)
```

Include file	<code>IoDrvDioClass1BitIn.h</code>						
Purpose	To get the values read from digital input channels.						
Description	<p>Because you get the values from the internal buffer of the driver object, there might be newer input states. To update the internal buffer with the latest states, you first have to call <code>DioCl1DigIn_read</code>.</p> <p>The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Bit I/O (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using <code>DioCl1DigIn_create</code>.</p> <p>pInputData Lets you specify the address of a variable that holds the bit values read from the digital inputs. The relevant bits are those bits of the least significant 16 bits which correspond to the channel mask specified when you created the driver object by using <code>DioCl1DigIn_create</code>. The most significant 16 bits are always set to zero.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						

DioCl1DigIn_getPortInData

Syntax	<pre>UInt32 DioCl1DigIn_getPortInData(DioCl1DigInSDrvObject *pDioCl1DigIn, UInt32 *pInputData)</pre>
---------------	---

Include file	<code>IoDrvDioClass1BitIn.h</code>						
Purpose	To get the values read from a digital input port.						
Description	<p>If you have specified to use all the channels of a port with your DIO Class 1 driver, you can use this function to get the values.</p> <p>Because you get the values from the internal buffer of the driver object, there might be newer input states. To update the internal buffer with the latest states, you first have to call <code>DioCl1DigIn_read</code>.</p> <p>The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Bit I/O (DIO Class 1 (MicroLabBox Features)).</p>						
Parameters	<p>pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using <code>DioCl1DigIn_create</code>.</p> <p>pInputData Lets you specify the address of a variable that holds the bit values read from the digital input port.</p> <p>The relevant bits are the least significant 16 bits (0x0000 ... 0xFFFF). The most significant 16 bits are always set to zero.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						

DioCl1DigIn_read

Syntax	<code>UInt32 DioCl1DigIn_read(DioCl1DigInSDrvObject *pDioCl1DigIn)</code>
---------------	---

Include file IoDrvDioClass1BitIn.h

Purpose To read data from the DIO Class 1 input channels.

Description This function is used to read the current states of the digital input channels specified when you created the DioCl1DigIn driver object by using **DioCl1DigIn_create**.

The channels must be enabled before by using **DioCl1DigIn_start**.

The channel states are stored in an internal buffer. To get the values, you must use one of the following functions.

- **DioCl1DigIn_getChMaskInData**, if you have specified only one channel or a subset of channels of the port to be used.
- **DioCl1DigIn_getPortInData**, if you have specified all the channels of the port to be used.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\)](#) ([MicroLabBox Features](#) ).

Parameters **pDioCl1DigIn** Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using **DioCl1DigIn_create**.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigIn_setEventDelay

Syntax

```
UInt32 DioCl1DigIn_setEventDelay(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    Float64 EventDelay)
```

Include file

`IoDrvDioClass1BitIn.h`

Purpose

To set the delay for events on a DIO Class 1 input channel.

Description

The function is used to delay the generation of events that you specified by using `DioCl1DigIn_setInterruptMode` and `DioCl1DigIn_setTriggerMode`. With these functions, you have also specified the channel that is affected by the delay setting.

The delay is the time interval between the occurrence of the event and the generation of the event signal (interrupt or trigger signal).

The setting does not take effect until you have called `DioCl1DigIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using `DioCl1DigIn_create`.

EventDelay Lets you specify the time interval by which the generation of an event is delayed in seconds in the range 0 ... 1.34 s. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

DioCl1DigIn_setEventDownSample

Syntax

```
UInt32 DioCl1DigIn_setEventDownsample(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    UInt32 DownsamplingValue)
```

Include file

`IoDrvDioClass1BitIn.h`

Purpose

To set the downsampling for events on a DIO Class 1 input channel.

Description

The function is used to reduce the number of generated events that you specified by using `DioCl1DigIn_setInterruptMode` and `DioCl1DigIn_setTriggerMode`. With these functions you have also specified the channel that is affected by the downsampling setting.

The setting does not take effect until you have called `DioCl1DigIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using `DioCl1DigIn_create`.

DownsamplingValue Lets you specify the downsampling value for event generation (interrupt or trigger signal) in the range 1 ... 256.

The downsampling value specifies whether to generate an event on each occurrence of the event condition (DownsamplingValue=1, default) or only on each n-th occurrence, where n is the specified downsampling value.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

DioCl1DigIn_setInputFilter

Syntax

```
UInt32 DioCl1DigIn_setInputFilter(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    Float64 FilterInterval)
```

Include file

`IoDrvDioClass1BitIn.h`

Purpose

To set the input filter of the DIO Class 1 unit.

Description

The input filter will be applied to all the channels that are controlled by the specified DioCl1DigIn driver object.

Only those input signals will be processed which have constant values over a longer time than specified by the `FilterInterval` parameter.

The setting does not take effect until you have called `DioCl1DigIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using `DioCl1DigIn_create`.

FilterInterval Lets you specify the minimum time interval for which the input signal(s) must provide a constant value that is to be used for processing. Shorter input signals will be filtered. You can specify filter intervals in seconds in the range 0 ... 0.01 s.

The resolution/deviation depends on the specified range:

- 0 ... 640 ns
Resolution of 10 ns
- >640 ns ... 0.01 s
Deviation of <10%

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigIn_setInterruptMode

Syntax

```
UInt32 DioCl1DigIn_setInterruptMode(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    UInt32 Channel
    UInt32 InterruptMode)
```

Include file

`IoDrvDioClass1BitIn.h`

Purpose

To set the interrupt mode of a DIO Class 1 input channel.

Description

This function is used to generate interrupts on rising, falling, or both edges of the input signal connected to the specified channel.

If you want to configure a combination of interrupts and trigger signals, you have to configure the trigger mode in addition to the interrupt mode by using `DioCl1DigIn_setTriggerMode`.

Note

You can specify only one channel for generating interrupts from the current DioCl1DigIn driver object.

The setting does not take effect until you have called `DioCl1DigIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1 \(MicroLabBox Features\)\)](#).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using **DioCl1DigIn_create**.

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

InterruptMode Lets you specify the interrupt mode. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_NO_INTERRUPT	No interrupt is to be generated (default).
DIO_CLASS1_INT_RISING_EDGE	Interrupts are generated on rising edges.
DIO_CLASS1_INT_FALLING_EDGE	Interrupts are generated on falling edges.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigIn_setIoIntVector

Syntax

```
UInt32 DioCl1DigIn_SetIoIntVector(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    UInt32 Channel,
    UInt32 EdgeSelect,
    IoLibTToIntHandler IoIntHandler)
```

Include file

IoDrvDioClass1BitIn.h

Purpose

To set the interrupt handler for processing interrupts from a DIO Class 1 input channel.

Description

This function is used to register the function that handles the generated interrupts specified by using **DioCl1DigIn_SetInterruptMode**. For each

specified condition for interrupt generation (rising edge, falling edge), you can define separate functions. If you want to handle all the conditions with the same function, you can use the bitwise OR operator to combine the predefined symbols of the **EdgeSelect** parameter.

Note

The condition for interrupt generation must also be specified for the **DioCl1DigIn_SetInterruptMode** function.

The setting does not take effect until you have called **DioCl1DigIn_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1 \(MicroLabBox Features\)\)](#).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using **DioCl1DigIn_create**.

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

EdgeSelect Lets you specify the edge the given interrupt handler is to be associated with. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_RISING_EDGE	The handler is to process interrupts generated on rising edges.
DIO_CLASS1_INT_FALLING_EDGE	The handler is to process interrupts generated on falling edges.

IoIntHandler Lets you specify the pointer to the function which is to handle the generated interrupt.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1DigIn_setTriggerLineOut

Syntax	<code>UInt32 DioCl1DigIn_setTriggerLineOut(DioCl1DigInSDrvObject *pDioCl1DigIn, UInt32 TriggerLineOut)</code>
---------------	--

Include file	<code>IoDrvDioClass1BitIn.h</code>
---------------------	------------------------------------

Purpose	To select the trigger line used for trigger signal generation.
----------------	--

Description	A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board. You must specify the trigger line the consumer is to listen to by using the relevant <code>xxx_setTriggerLineIn</code> function. A trigger line can be allocated only once, but multiple consumers can listen to it.
--------------------	--

Before you can use trigger signals, you have to set the trigger mode by using `DioCl1DigIn_setTriggerMode`. With `DioCl1DigIn_setTriggerLineOutStatus` you enable or disable the trigger line.

The setting does not take effect until you have called `DioCl1DigIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) .
--------------------	---

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1 \(MicroLabBox Features\)\)](#).

Parameters	pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using <code>DioCl1DigIn_create</code> .
-------------------	---

TriggerLineOut Lets you specify the number of the trigger line to be used for transmitting the trigger signal in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.
...	...
DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

[DioCl1DigIn_setTriggerLineOutStatus](#)..... 285

DioCl1DigIn_setTriggerLineOutStatus

Syntax

```
UInt32 DioCl1DigIn_setTriggerLineOutStatus(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    UInt32 Status)
```

Include file

`IoDrvDioClass1BitIn.h`

Purpose

To set the status of the trigger line.

Description

With this function, you can enable or disable the trigger line that you specified before by using **DioCl1DigIn_setTriggerLineOut**.

The function is intended to be called during the initialization phase of the real-time application or during a run-to-stop or a stop-to-run transition of the real-time application.

The setting does not take effect until you call **DioCl1DigIn_start**.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using [DioCl1DigIn_create](#).

Status Lets you specify the status of the trigger line.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_DISABLE	Disables the trigger line.
DIO_CLASS1_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1DigIn_create.....	270
DioCl1DigIn_start.....	288

DioCl1DigIn_setTriggerMode

Syntax

```
UInt32 DioCl1DigIn_setTriggerMode(
    DioCl1DigInSDrvObject *pDioCl1DigIn,
    UInt32 Channel,
    UInt32 TriggerMode)
```

Include file

[IoDrvDioClass1BitIn.h](#)

Purpose

To set the trigger mode of a DIO Class 1 input channel.

Description

This function is used to generate trigger signals on rising, falling, or both edges of the input signal connected to the specified channel. Additionally, you have to specify the trigger line for the generated trigger signal by using [DioCl1DigIn_setTriggerLineOut](#) on page 284. The I/O unit that is to consume the trigger signal has to specify this trigger line by using its `xxx_SetTriggerLineIn` function.

If you want to configure a combination of interrupts and trigger signals, you have to configure the interrupt mode in addition to the trigger mode by using [DioCl1DigIn_SetInterruptMode](#).

Note

You can specify only one channel for generating trigger signals from the current `DigCl1DigIn` driver object.

The setting does not take effect until you have called [DioCl1DigIn_apply](#) during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1 \(MicroLabBox Features\)\)](#).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The `DioCl1DigIn` driver object must already have been created by using [DioCl1DigIn_create](#).

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

TriggerMode Lets you specify the trigger mode. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_NO_TRIGGER	No trigger is to be generated (default).
DIO_CLASS1_TRIG_RISING_EDGE	Triggers are generated on rising edges.
DIO_CLASS1_TRIG_FALLING_EDGE	Triggers are generated on falling edges.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigIn_start

Syntax

```
UInt32 DioCl1DigIn_start(DioCl1DigInSDrvObject *pDioCl1DigIn)
```

Include file

IoDrvDioClass1BitIn.h

Purpose

To start the DIO Class 1 input channels.

Description

This function starts the specified DioCl1DigIn driver so that it will activate its digital input channels. The digital input channels are enabled to sample the input signals. Afterwards, the channel states can be read. This function also enables the configured trigger line. If you call `DioCl1DigIn_setTriggerLineOutStatus(DIO_CLASS1_TRIGGER_LINE_DISABLE)` before, the trigger line stays disabled.

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigIn Lets you specify the DIO Class 1 channels to be used. The DioCl1DigIn driver object must already have been created by using `DioCl1DigIn_create`.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

[DioCl1DigIn_setTriggerLineOutStatus](#)..... 285

Example of Implementing Bit I/O Inputs Using DIO Class 1

Introduction

The following example code shows you the order in which the functions are to be used. The code cannot be built without modifications.

Using DIO Class 1 input channels

The following code shows you how to use channels 1 and 2 for signal measurement by using the initial configuration.

```
DioCl1DigInSDrvObject* MyBITINCL1;
UInt32 IErr;
UInt32 Data;
...
/* Initialize the DIO Class 1 unit.*/
/* Create the driver object that is using channels 1 and 2 of port 1 of the DIO Class 1 unit.*/
IErr=DioCl1DigIn_create
    (&MyBITINCL1, DIO_CLASS1_PORT_1, DIO_CLASS1_MASK_CH_1|DIO_CLASS1_MASK_CH_2);
/* Apply the default settings to the driver object.*/
IErr=DioCl1DigIn_apply(MyBITINCL1);
/* Enable the digital input channels.*/
IErr=DioCl1DigIn_start(MyBITINCL1);
/* Read the current channel states from the channel.*/
IErr=DioCl1DigIn_read(MyBITINCL1);
/* Get the measured value from the internal buffer.*/
IErr=DioCl1DigIn_getChMaskInData(MyBITINCL1, &Data);
...
```

The following code shows you how to use all the channels from port 2 for signal measurement. The input signals will be filtered. Additionally, channel 1 is used to generate an interrupt and a trigger signal on each second rising edge with a delay of 100 ns.

```

DioCl1DigInSDrvObject* MyBITINCL1;
UInt32 IErr;
UInt32 Data;
...
/* Initialize the DIO Class 1 unit. */
/* Create the driver object that is using all the channels of port 2 of the DIO Class 1 unit. */
IErr=DioCl1DigIn_create
    (&MyBITINCL1, DIO_CLASS1_PORT_2, 0xFFFF);
/* Set the input filter to 600 ns. */
IErr=DioCl1DigIn_setInputFilter(MyBITINCL1,0.0000006);
/* Set the interrupt mode. */
IErr=DioCl1DigIn_setInterruptMode
    (MyBITINCL1, DIO_CLASS1_CHANNEL_1, DIO_CLASS1_INT_RISING_EDGE);
/* Register the interrupt handler function. */
IErr=DioCl1DigIn_setIoIntVector
    (MyBITINCL1, DIO_CLASS1_CHANNEL_1, DIO_CLASS1_INT_RISING_EDGE, MyInterruptFunction);
/* Set the trigger mode. */
IErr=DioCl1DigIn_setTriggerMode
    (MyBITINCL1, DIO_CLASS1_CHANNEL_1, DIO_CLASS1_TRIG_RISING_EDGE);
/* Set the trigger line to be used for transmitting the generated trigger signal. */
IErr=DioCl1DigIn_setTriggerLineOut
    (MyBITINCL1, DIO_CLASS1_TRIGGER_LINE_1);
/* Set the downsample value for interrupts and trigger signals (each second event). */
IErr=DioCl1DigIn_setEventDownSample
    (MyBITINCL1, 2);
/* Set the delay value for interrupts and trigger signals (100 µs). */
IErr=DioCl1DigIn_setEventDelay
    (MyBITINCL1, 0.0001);
/* Apply the initialization settings to the driver object. */
IErr=DioCl1DigIn_apply(MyBITINCL1);
/* Enable the digital input channels. */
IErr=DioCl1DigIn_start(MyBITINCL1);
/* Enable interrupt generation and write it to the driver. */
IErr=DioCl1DigIn_enableInterrupts(MyBITINCL1, DIO_CLASS1_INT_RISING_EDGE);
/* Read the current channel states from the port. */
IErr=DioCl1DigIn_read(MyBITINCL1);
/* Get the measured value from the internal buffer. */
IErr=DioCl1DigIn_getPortInData(MyBITINCL1, &Data);
...

```

Digital Out Class 1

Introduction

The Digital Out Class 1 unit can be accessed via the **DioCl1DigOut** functions.

Where to go from here

Information in this section

DioCl1DigOut_apply To apply the initialization settings to the DIO Class 1 output channels.	292
--	-----

DioCl1DigOut_create	293
To create the I/O driver object for digital signal outputs for bit I/O via the DIO Class 1 unit.	
DioCl1DigOut_disableInterrupts	294
To disable the generation of interrupts on a DIO Class 1 output channel.	
DioCl1DigOut_enableInterrupts	295
To enable the generation of interrupts on a DIO Class 1 output channel.	
DioCl1DigOut_setChMaskOutData	297
To set the values to be output by the specified DIO Class 1 output channels.	
DioCl1DigOut_setChMaskOutHighZ	298
To set the high impedance state of DIO Class 1 output channels.	
DioCl1DigOut_setEventDelay	299
To set the delay for events on a DIO Class 1 output channel.	
DioCl1DigOut_setEventDownsample	300
To set the downsampling for events on a DIO Class 1 output channel.	
DioCl1DigOut_setInterruptMode	301
To set the interrupt mode of a DIO Class 1 output channel.	
DioCl1DigOut_setIoIntVector	303
To set the interrupt handler for processing interrupts from a DIO Class 1 output channel.	
DioCl1DigOut_setPortOutData	304
To set the values to be written to a DIO Class 1 output port.	
DioCl1DigOut_setPortOutHighZ	305
To set the high impedance state of a DIO Class 1 output port.	
DioCl1DigOut_setRisingEdgeDelay	307
To set the time for delaying the rising edges of the DIO Class 1 output channels.	
DioCl1DigOut_setSignalVoltage	308
To set the voltage level used for the DIO Class 1 output signals.	
DioCl1DigOut_setTriggerLineOut	309
To select the trigger line used for trigger signal generation.	
DioCl1DigOut_setTriggerLineOutStatus	310
To set the status of the trigger line.	
DioCl1DigOut_setTriggerMode	311
To set the trigger mode of a DIO Class 1 output channel.	
DioCl1DigOut_start	313
To start the DIO Class 1 output channels.	
DioCl1DigOut_write	314
To update settings of the specified DIO Class 1 output channels during run time.	

Example of Implementing Bit I/O Outputs Using DIO Class 1.....315

Shows you how to implement access to the digital output channels of the DIO Class 1 unit.

Information in other sections

[Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#)

DioCl1DigOut_apply

Syntax

```
UInt32 DioCl1DigOut_apply(DioCl1DigOutSDrvObject *pDioCl1DigOut)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To apply the initialization settings to the DIO Class 1 output channels.

Description

Before you can start the specified DIO Class 1 channels, you have to transfer the settings of the DioCl1DigOut driver object to its related output channels by using `DioCl1DigOut_apply`. You have to do this also for the default values and for values that you specified by using the `DioCl1DigOut_set<Parameter>` functions. The `DioCl1DigOut_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The signal generation on the specified digital output channel(s) is not activated until you have called `DioCl1DigOut_start`.

To update settings during run time, you have to use `DioCl1DigOut_write`.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using `DioCl1DigOut_create`.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_create

Syntax

```
UInt32 DioCl1DigOut_create(
    DioCl1DigOutSDrvObject **ppDioCl1DigOut,
    UInt32 Port,
    UInt32 ChannelMask)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To create the I/O driver object for digital signal outputs for bit I/O via the DIO Class 1 unit.

Description

This function performs all the steps necessary to create an I/O driver object for the bit I/O of the DIO Class 1 unit. The 48 digital channels of the board are separated into three ports with 16 channels each. A DioCl1DigOut driver object can only handle the channels of one port. Via a channel mask, you can therefore specify up to 16 channels to be used with the current driver object.

The specified DIO Class 1 channels are initialized with default values so that they are ready for use.

The initial values are:

- Output signal voltage level: 2.5 V
- Rising edge delay: 0
- Interrupt mode: no interrupt generation
- Trigger mode: no trigger signal generation
- Event downsample: 1
- Event delay: 0 s

The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

ppDioCl1DigOut Lets you specify the address of a variable which is to hold the address of the driver object created.

Port Lets you specify the number of the DIO Class 1 port providing the required channels in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
DIO_CLASS1_PORT_2	Specifies port 2 of the DIO Class 1 unit.
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

ChannelMask Lets you specify the channel(s) of the specified port to be controlled by the driver in the range 1 ... 16. To specify more than one channel, you can combine the symbols by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_MASK_CH_1	Specifies channel 1 of the DIO Class 1 unit.
...	...
DIO_CLASS1_MASK_CH_16	Specifies channel 16 of the DIO Class 1 unit.

Tip

To specify all the 16 channels of a port, you can enter `0xFFFF` as the channel mask.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_disableInterrupts

Syntax

```
UInt32 DioCl1DigOut_disableInterrupts(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 InterruptSelect)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose	To disable the generation of interrupts on a DIO Class 1 output channel.						
Description	<p>With this function, you can disable the generation of interrupts that you specified by using DioCl1DigOut_setInterruptMode. The settings take effect immediately.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Bit I/O (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using DioCl1DigOut_create.</p> <p>InterruptSelect Lets you select the interrupts that are to be disabled. The symbols can be combined by using the bitwise OR operator.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_INT_RISING_EDGE</td><td>Disables the generation of interrupts on rising edges.</td></tr> <tr> <td>DIO_CLASS1_INT_FALLING_EDGE</td><td>Disables the generation of interrupts on falling edges.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_INT_RISING_EDGE	Disables the generation of interrupts on rising edges.	DIO_CLASS1_INT_FALLING_EDGE	Disables the generation of interrupts on falling edges.
Symbol	Meaning						
DIO_CLASS1_INT_RISING_EDGE	Disables the generation of interrupts on rising edges.						
DIO_CLASS1_INT_FALLING_EDGE	Disables the generation of interrupts on falling edges.						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1DigOut_enableInterrupts

Syntax

```
UInt32 DioCl1DigOut_enableInterrupts(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 InterruptSelect)
```

Include file	<code>IoDrvDioClass1BitOut.h</code>						
Purpose	To enable the generation of interrupts on a DIO Class 1 output channel.						
Description	<p>With this function, you can either enable the generation of interrupts or reenable the generation of interrupts that you disabled before by using <code>DioCl1DigOut_disableInterrupts</code>. The interrupts have to be specified before by using <code>DioCl1DigOut_setInterruptMode</code>. The settings take effect immediately.</p> <p>After calling <code>DioCl1DigOut_start</code>, interrupts are disabled.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Bit I/O (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using <code>DioCl1DigOut_create</code>.</p> <p>InterruptSelect Lets you select the interrupts that are to be enabled. The symbols can be combined by using the bitwise OR operator.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>DIO_CLASS1_INT_RISING_EDGE</code></td><td>Enables the generation of interrupts on rising edges.</td></tr> <tr> <td><code>DIO_CLASS1_INT_FALLING_EDGE</code></td><td>Enables the generation of interrupts on falling edges.</td></tr> </tbody> </table>	Symbol	Meaning	<code>DIO_CLASS1_INT_RISING_EDGE</code>	Enables the generation of interrupts on rising edges.	<code>DIO_CLASS1_INT_FALLING_EDGE</code>	Enables the generation of interrupts on falling edges.
Symbol	Meaning						
<code>DIO_CLASS1_INT_RISING_EDGE</code>	Enables the generation of interrupts on rising edges.						
<code>DIO_CLASS1_INT_FALLING_EDGE</code>	Enables the generation of interrupts on falling edges.						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						

Related topics

References

DioCl1DigOut_disableInterrupts.....	294
DioCl1DigOut_setInterruptMode.....	301

DioCl1DigOut_setChMaskOutData

Syntax

```
UInt32 DioCl1DigOut_setChMaskOutData(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 OutputData)
```

Include file

IoDrvDioClass1BitOut.h

Purpose

To set the values to be output by the specified DIO Class 1 output channels.

Description

The specified output data is initially stored in an internal buffer of the driver object and is not output.

The setting does not take effect until you call **DioCl1DigOut_apply** during the initialization phase or **DioCl1DigOut_write** during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the outputs that you specified by using **DioCl1DigOut_create**, you must use **DioCl1DigOut_start**.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using **DioCl1DigOut_create**.

OutputData Lets you specify the bit values to be output on the digital outputs. The relevant bits are those bits of the least significant 16 bits which correspond to the channel mask specified when you created the driver object by using **DioCl1DigOut_create**. The most significant 16 bits of the parameter are ignored.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_setChMaskOutHighZ

Syntax

```
UInt32 DioCl1DigOut_setChMaskOutHighZ(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 HighZStateOnOff)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To set the high impedance state of DIO Class 1 output channels.

Description

You can force the output channels of the driver object to the high impedance state, for example, for termination purposes, by setting the high impedance state to *On*. If you want the real-time application to control the output levels, you can release the high impedance state again.

The setting takes effect only after you call **DioCl1DigOut_start** or **DioCl1DigOut_write** during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using **DioCl1DigOut_create**.

HighZStateOnOff Lets you specify if outputs are to be set to the high impedance state or if outputs are to be released from the high impedance state.

Symbol	Meaning
DIO_CLASS1_HIGH_Z_OFF	Releases the high impedance state.
DIO_CLASS1_HIGH_Z_ON	Sets outputs to the high impedance state (default).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1DigOut_start.....	313
DioCl1DigOut_write.....	314

DioCl1DigOut_setEventDelay

Syntax

```
UInt32 DioCl1DigOut_setEventDelay(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    Float64 EventDelay)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To set the delay for events on a DIO Class 1 output channel.

Description

The function is used to delay the generation of events that you specified by using **DioCl1DigOut_setInterruptMode** and **DioCl1DigOut_setTriggerMode**. With these functions you have also specified the channel that is affected by the delay setting.

The delay is the time interval between the occurrence of the event and the generation of the event signal (interrupt or trigger signal).

The setting does not take effect until you have called **DioCl1DigOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1 \(MicroLabBox Features\)\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using **DioCl1DigOut_create**.

EventDelay Lets you specify the time interval by which the generation of an event is delayed in seconds in the range 0 ... 1.34 s. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_setEventDownsample

Syntax

```
UInt32 DioCl1DigOut_setEventDownsample(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 DownSamplingValue)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To set the downsampling for events on a DIO Class 1 output channel.

Description

The function is used to reduce the number of generated events that you specified by using **DioCl1DigOut_setInterruptMode** and **DioCl1DigOut_setTriggerMode**. With these functions you have also specified the channel that is affected by the downsampling setting.

The setting does not take effect until you have called [DioCl1DigOut_apply](#) during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using [DioCl1DigOut_create](#).

DownsamplingValue Lets you specify the downsampling value for event generation (interrupt or trigger signal) in the range 1 ... 256.

The downsampling value specifies whether to generate an event on each occurrence of the event condition (DownsamplingValue=1, default) or only on each n-th occurrence, where n is the specified downsampling value.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_setInterruptMode

Syntax

```
UInt32 DioCl1DigOut_setInterruptMode(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 Channel,
    UInt32 InterruptMode)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To set the interrupt mode of a DIO Class 1 output channel.

Description This function is used to generate interrupts on rising, falling or both edges of the output signal connected to the specified channel.

If you want to configure a combination of interrupts and trigger signals, you have to configure the trigger mode in addition to the interrupt mode by using **DioCl1DigOut_setTriggerMode**.

Note

You can specify only one channel from the current **DigCl1DigOut** driver object for generating interrupts.

The setting does not take effect until you have called **DioCl1DigOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters **pDioCl1DigOut** Lets you specify the DIO Class 1 channels to be used. The **DioCl1DigOut** driver object had to be created before by using **DioCl1DigOut_create**.

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

InterruptMode Lets you specify the interrupt mode. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_NO_INTERRUPT	No interrupt is to be generated (default).
DIO_CLASS1_INT_RISING_EDGE	Interrupts are generated on rising edges.
DIO_CLASS1_INT_FALLING_EDGE	Interrupts are generated on falling edges.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_setIoIntVector

Syntax

```
UInt32 DioCl1DigOut_setIoIntVector(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 Channel,
    UInt32 EdgeSelect,
    IolibTToIntHandler IoIntHandler)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To set the interrupt handler for processing interrupts from a DIO Class 1 output channel.

Description

This function is used to register the function that handles the generated interrupts specified by using `DioCl1DigOut_setInterruptMode`. For each specified condition for generating an interrupt (rising edge, falling edge), you can define separate functions. If you want to handle all the conditions with the same function, you can use the bitwise OR operator to combine the predefined symbols of the `EdgeSelect` parameter.

Note

The condition for interrupt generation must be the same as specified for the `DioCl1DigOut_setInterruptMode` function.

The setting does not take effect until you have called `DioCl1DigOut_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using **DioCl1DigOut_create**.

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

EdgeSelect Lets you specify the edge the given interrupt handler is to be associated with. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_RISING_EDGE	The handler is to process interrupts generated on rising edges.
DIO_CLASS1_INT_FALLING_EDGE	The handler is to process interrupts generated on falling edges.

IoIntHandler Lets you specify the pointer to the function which is to handle the generated interrupt.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_setPortOutData

Syntax

```
UInt32 DioCl1DigOut_setPortOutData(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 OutputData)
```

Include file

IoDrvDioClass1BitOut.h

Purpose

To set the values to be written to a DIO Class 1 output port.

Description

If you have specified to use all the channels of a port with your DIO Class 1 driver, you can use this function to set the values.

Because you initially write the values to the internal buffer of the driver object, you have to explicitly start the output afterwards.

The setting does not take effect until you call `DioCl1DigOut_apply` during the initialization phase or `DioCl1DigOut_write` during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the outputs that you specified by using `DioCl1DigOut_create`, you must use `DioCl1DigOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using `DioCl1DigOut_create`.

OutputData Lets you specify the bit values to be output on the digital output port. The relevant bits are those bits of the least significant 16 bits which correspond to the channel mask `0xFFFF` specified when you created the driver object. The most significant 16 bits of the parameter are ignored.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_setPortOutHighZ

Syntax

```
UInt32 DioCl1DigOut_setPortOutHighZ(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 HighZStateOnOff)
```

Include file	IoDrvDioClass1BitOut.h						
Purpose	To set the high impedance state of a DIO Class 1 output port.						
Description	<p>If you have specified the channel mask 0xFFFF when you created the DioCl1DigOut driver object, you can use this function to set the high impedance state for all the channels of the port.</p> <p>You can force the output channels of the driver object to the high impedance state, for example, for termination purposes, by setting the high impedance state to <i>On</i>. If you want the real-time application to control the output levels, you can release the high impedance state again.</p> <p>The setting takes effect only after you call DioCl1DigOut_start or DioCl1DigOut_write during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Bit I/O (DIO Class 1 (MicroLabBox Features)).</p>						
Parameters	<p>pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using DioCl1DigOut_create.</p> <p>HighZStateOnOff Lets you specify if outputs are to be set to the high impedance state or if outputs are to be released from the high impedance state.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_HIGH_Z_OFF</td><td>Releases the high impedance state.</td></tr> <tr> <td>DIO_CLASS1_HIGH_Z_ON</td><td>Sets outputs to the high impedance state (default).</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_HIGH_Z_OFF	Releases the high impedance state.	DIO_CLASS1_HIGH_Z_ON	Sets outputs to the high impedance state (default).
Symbol	Meaning						
DIO_CLASS1_HIGH_Z_OFF	Releases the high impedance state.						
DIO_CLASS1_HIGH_Z_ON	Sets outputs to the high impedance state (default).						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1DigOut_setRisingEdgeDelay

Syntax

```
UInt32 DioCl1DigOut_setRisingEdgeDelay(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    Float64 RisingEdgeDelay)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To set the time for delaying the rising edges of the DIO Class 1 output channels.

Description

The delay will be applied to all the channels that are controlled by the specified DioCl1DigOut driver object.

The generated rising edges of the output signals will be output after the specified time.

The setting does not take effect until you have called `DioCl1DigOut_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using `DioCl1DigOut_create`.

RisingEdgeDelay Lets you specify the delay applied to the rising edges of the output signal in seconds in the range 0 ... 655 µs. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_setSignalVoltage

Syntax

```
UInt32 DioCl1DigOut_setSignalVoltage(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 SignalVoltageSelector)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To set the voltage level used for the DIO Class 1 output signals.

Description

The output voltage level will be applied to all the channels that are controlled by the specified DioCl1DigOut driver object.

The setting does not take effect until you have called **DioCl1DigOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1 \(MicroLabBox Features\)\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using **DioCl1DigOut_create**.

SignalVoltageSelector Lets you select the voltage level used for the output signal(s).

Symbol	Meaning
DIO_CLASS1_SIGNAL_2_5_V	Specifies 2.5 V as the output voltage level (default).
DIO_CLASS1_SIGNAL_3_3_V	Specifies 3.3 V as the output voltage level.
DIO_CLASS1_SIGNAL_5_0_V	Specifies 5.0 V as the output voltage level.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_setTriggerLineOut

Syntax

```
UInt32 DioCl1DigOut_setTriggerLineOut(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 TriggerLineOut)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To select the trigger line used for trigger signal generation.

Description

A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board. You have to specify the trigger line the consumer is to listen to by using the relevant `xxx_setTriggerLineIn` function. A trigger line can be allocated only once, but multiple consumers can listen to it.

Before you can use trigger signals, you have to set the trigger mode by using `DioCl1DigOut_setTriggerMode`. With `DioCl1DigOut_setTriggerLineOutStatus` you enable or disable the trigger line.

The setting does not take effect until you have called `DioCl1DigOut_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using `DioCl1DigOut_create`.

TriggerLineOut Lets you specify the number of the trigger line to be used for transmitting the trigger signal in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.
...	...
DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1DigOut_setTriggerLineOutStatus.....	310
DioCl1DigOut_setTriggerMode.....	311

DioCl1DigOut_setTriggerLineOutStatus

Syntax

```
UInt32 DioCl1DigOut_setTriggerLineOutStatus(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 Status)
```

Include file

IoDrvDioClass1BitOut.h

Purpose

To set the status of the trigger line.

Description

With this function, you can enable or disable the trigger line that you specified before by using [DioCl1DigOut_setTriggerLineOut](#).

The function is intended to be called during the initialization phase of the real-time application or during a run-to-stop or a stop-to-run transition of the real-time application.

The setting does not take effect until you call [DioCl1DigOut_start](#).

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using [DioCl1DigOut_create](#).

Status Lets you specify the status of the trigger line.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_DISABLE	Disables the trigger line.
DIO_CLASS1_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1DigOut_create.....	293
DioCl1DigOut_start.....	313

DioCl1DigOut_setTriggerMode

Syntax

```
UInt32 DioCl1DigOut_setTriggerMode(
    DioCl1DigOutSDrvObject *pDioCl1DigOut,
    UInt32 Channel,
    UInt32 TriggerMode)
```

Include file

[IoDrvDioClass1BitOut.h](#)

Purpose

To set the trigger mode of a DIO Class 1 output channel.

Description

This function is used to generate trigger signals on rising, falling or both edges of the output signal connected to the specified channel. Additionally, you have to specify the trigger line for the generated trigger signal by using **DioCl1DigOut_setTriggerLineOut**. The I/O unit that is to consume the trigger signal has to specify this trigger line by using its associated **xxx_setTriggerLineIn** function.

If you want to configure a combination of interrupts and trigger signals, you have to configure the interrupt mode in addition to the trigger mode by using **DioCl1DigOut_setInterruptMode**.

Note

You can specify only one channel from the current **DigCl1DigOut** driver object for generating trigger signals.

The setting does not take effect until you have called **DioCl1DigOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1 \(MicroLabBox Features\)\)](#).

Parameters

pDioCl1DigOut Lets you specify the DIO Class 1 channels to be used. The **DioCl1DigOut** driver object had to be created before by using **DioCl1DigOut_create**.

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

TriggerMode Lets you specify the trigger mode. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_NO_TRIGGER	No trigger is to be generated (default).
DIO_CLASS1_TRIG_RISING_EDGE	Triggers are generated on rising edges.
DIO_CLASS1_TRIG_FALLING_EDGE	Triggers are generated on falling edges.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1DigOut_start

Syntax

```
UInt32 DioCl1DigOut_start(DioCl1DigOutSDrvObject *pDioCl1DigOut)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To start the DIO Class 1 output channels.

Description

This function starts the specified DioCl1DigOut driver so that it will activate its digital output channels. The digital output channels are enabled to generate the output signals. If you call `DioCl1DigOut_setPortOutHighZ(DIO_CLASS1_HIGH_Z_ON)` or `DioCl1DigOut_setChMaskOutHighZ(DIO_CLASS1_HIGH_Z_ON)` before, the output channels stay in the high impedance state.

`DioCl1DigOut_start` also enables the trigger line. If you call `DioCl1DigOut_setTriggerLineOutStatus(DIO_CLASS1_TRIGGER_LINE_DISABLE)` before, the trigger line stays disabled.

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

`pDioCl1DigOut` Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using `DioCl1DigOut_create`.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1DigOut_setChMaskOutHighZ.....	298
DioCl1DigOut_setPortOutHighZ.....	305
DioCl1DigOut_setTriggerLineOutStatus.....	310

DioCl1DigOut_write

Syntax

```
UInt32 DioCl1DigOut_write(DioCl1DigOutSDrvObject *pDioCl1DigOut)
```

Include file

`IoDrvDioClass1BitOut.h`

Purpose

To update settings of the specified DIO Class 1 output channels during run time.

Description

If have used one of the following functions during run time, their settings will not be updated on the hardware until you called `DioCl1DigOut_write`.

- `DioCl1DigOut_setChMaskOutData`
- `DioCl1DigOut_setPortOutData`
- `DioCl1DigOut_setChMaskOutHighZ`
- `DioCl1DigOut_setPortOutHighZ`

The specified digital output channel(s) must be enabled by using `DioCl1DigOut_start`.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Bit I/O \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters **pDioCl1DigOut** Lets you specify the DIO Class 1 channels to be used. The DioCl1DigOut driver object had to be created before by using **DioCl1DigOut_create**.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Example of Implementing Bit I/O Outputs Using DIO Class 1

Introduction The following example code shows you in which order the functions are to be used. The code cannot be built without modifications.

Using DIO Class 1 output channels The following code shows you how to use channels 1 and 2 from port 1 for signal generation using the initial configuration.

```

DioCl1DigOutSDrvObject* MyBITOUTCL1;
UInt32 IErr;
UInt32 Data = 0x3;
...
/* Initialize the DIO Class 1 unit.*/
/* Create the driver object that is using channels 1 and 2 of port 1 of the DIO Class 1 unit.*/
IErr=DioCl1DigOut_create
    (&MyBITOUTCL1, DIO_CLASS1_PORT_1, DIO_CLASS1_MASK_CH_1|DIO_CLASS1_MASK_CH_2);
/* Apply the default settings to the driver object.*/
IErr=DioCl1DigOut_apply(MyBITOUTCL1);
/* Enable the digital output channels.*/
IErr=DioCl1DigOut_start(MyBITOUTCL1);
/* Write the output states to the internal buffer.*/
IErr=DioCl1DigOut_setChMaskOutData(MyBITOUTCL1, Data);
/* Output the data at the output channels.*/
IErr=DioCl1DigOut_write(MyBITOUTCL1);
...

```

The following code shows you how to use all the channels from port 2 for signal generation. The output signals will be output with a level of 5 V and a rising

edge delay of 100 ns. Additionally, channel 1 is used to generate an interrupt and a trigger signal on each second rising edge with a delay of 50 ns.

```

DioCl1DigOutSDrvObject* MyBITOUTCL1;
UInt32 IErr;
UInt32 Data = 0xFFFF;

...
/* Initialize the DIO Class 1 unit. */
/* Create the driver object that is using all the channels of port 2 of the DIO Class 1 unit. */
IErr=DioCl1DigOut_create(
    &MyBITOUTCL1, DIO_CLASS1_PORT_2, 0xFFFF);
/* Set the edge delay to 100 ns. */
IErr=DioCl1DigOut_setRisingEdgeDelay(
    MyBITOUTCL1,0.000001);
/* Set the output voltage to 5 V. */
IErr=DioCl1DigOut_setSignalVoltage(
    MyBITOUTCL1, DIO_CLASS1_SIGNAL_5_0_V);
/* Set the interrupt mode. */
IErr=DioCl1DigOut_setInterruptMode(
    MyBITOUTCL1, DIO_CLASS1_CHANNEL_1, DIO_CLASS1_INT_RISING_EDGE);
/* Register the interrupt handler function. */
IErr=DioCl1DigOut_setToIntVector(
    MyBITOUTCL1, DIO_CLASS1_CHANNEL_1, DIO_CLASS1_INT_RISING_EDGE, MyInterruptFunction);
/* Set the trigger mode. */
IErr=DioCl1DigOut_setTriggerMode(
    MyBITOUTCL1, DIO_CLASS1_CHANNEL_1, DIO_CLASS1_TRIG_RISING_EDGE);
/* Set the trigger line to be used for transmitting the generated trigger signal. */
IErr=DioCl1DigOut_setTriggerLineOut(
    MyBITOUTCL1, DIO_CLASS1_TRIGGER_LINE_1);
/* Set the downsample value for interrupts and trigger signals (each second event). */
IErr=DioCl1DigOut_setEventDownsample(
    MyBITOUTCL1, 2);
/* Set the delay value for interrupts and trigger signals (50 ns). */
IErr=DioCl1DigOut_setEventDelay(
    MyBITOUTCL1, 0.0000005)
/* Apply the initialization settings to the driver object. */
IErr=DioCl1DigOut_apply(MyBITOUTCL1);
/* Enable the digital output channels. */
IErr=DioCl1DigOut_start(MyBITOUTCL1);
/* Enable interrupt generation and write it to the driver. */
IErr=DioCl1DigOut_enableInterrupts(
    MyBITOUTCL1, DIO_CLASS1_INT_RISING_EDGE);
/* Write the output states to the internal buffer. */
IErr=DioCl1DigOut_setPortOutData(
    MyBITOUTCL1, Data);
/* Output the data to the channels of the port. */
IErr=DioCl1DigOut_write(MyBITOUTCL1);
...

```

Timing I/O

Introduction MicroLabBox provides timing I/O functions for generating and measuring PWM signals and pulses.

Where to go from here	Information in this section								
	<table> <tr> <td>PWM Signal Generation (PWM Out).....</td> <td>317</td> </tr> <tr> <td>PWM Signal Measurement (PWM In).....</td> <td>342</td> </tr> <tr> <td>Pulse Signal Generation (Pulse Out).....</td> <td>364</td> </tr> <tr> <td>Pulse Width Measurement (Pulse In).....</td> <td>376</td> </tr> </table>	PWM Signal Generation (PWM Out).....	317	PWM Signal Measurement (PWM In).....	342	Pulse Signal Generation (Pulse Out).....	364	Pulse Width Measurement (Pulse In).....	376
PWM Signal Generation (PWM Out).....	317								
PWM Signal Measurement (PWM In).....	342								
Pulse Signal Generation (Pulse Out).....	364								
Pulse Width Measurement (Pulse In).....	376								
	Information in other sections								
	<table> <tr> <td>Implementing I/O Functions.....</td> <td>198</td> </tr> </table>	Implementing I/O Functions.....	198						
Implementing I/O Functions.....	198								

PWM Signal Generation (PWM Out)

Introduction MicroLabBox provides functions to generate PWM signals.

For details, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Where to go from here	Information in this section												
	<table> <tr> <td>DioCl1PwmOut_apply.....</td> <td>319</td> </tr> <tr> <td>To apply the initialization settings to the DIO Class 1 PWM output channels.</td> <td></td> </tr> <tr> <td>DioCl1PwmOut_create.....</td> <td>320</td> </tr> <tr> <td>To create the I/O driver object for PWM signal generation using a DIO Class 1 output channel.</td> <td></td> </tr> <tr> <td>DioCl1PwmOut_disableInterrupts.....</td> <td>321</td> </tr> <tr> <td>To disable the generation of interrupts on a DIO Class 1 PWM output channel.</td> <td></td> </tr> </table>	DioCl1PwmOut_apply.....	319	To apply the initialization settings to the DIO Class 1 PWM output channels.		DioCl1PwmOut_create.....	320	To create the I/O driver object for PWM signal generation using a DIO Class 1 output channel.		DioCl1PwmOut_disableInterrupts.....	321	To disable the generation of interrupts on a DIO Class 1 PWM output channel.	
DioCl1PwmOut_apply.....	319												
To apply the initialization settings to the DIO Class 1 PWM output channels.													
DioCl1PwmOut_create.....	320												
To create the I/O driver object for PWM signal generation using a DIO Class 1 output channel.													
DioCl1PwmOut_disableInterrupts.....	321												
To disable the generation of interrupts on a DIO Class 1 PWM output channel.													

DioCl1PwmOut_enableInterrupts	322
To enable the generation of interrupts on a DIO Class 1 PWM output channel.	
DioCl1PwmOut_setDutyCycle	324
To set the duty cycle of the PWM signal to be output at a DIO Class 1 channel.	
DioCl1PwmOut_setEventDelay	325
To set the delay for events on a DIO Class 1 output channel.	
DioCl1PwmOut_setEventDownsample	326
To set the downsampling for events on a DIO Class 1 output channel.	
DioCl1PwmOut_setInterruptMode	327
To set the interrupt mode of a DIO Class 1 output channel.	
DioCl1PwmOut_setInvertingMode	328
To set the inverting mode of the DIO Class 1 PWM output channel.	
DioCl1PwmOut_setToIntVector	329
To set the interrupt handler for processing interrupts from a DIO Class 1 PWM output channel.	
DioCl1PwmOut_setOutputHighZ	330
To set the high impedance state of the DIO Class 1 PWM output channel.	
DioCl1PwmOut_setPeriod	332
To set the period of the PWM signal to be output at a DIO Class 1 channel.	
DioCl1PwmOut_setRisingEdgeDelay	333
To set the time for delaying the rising edges of the DIO Class 1 PWM output channel.	
DioCl1PwmOut_setSignalVoltage	334
To set the voltage level used for the DIO Class 1 PWM output channel.	
DioCl1PwmOut_setTriggerLineOut	335
To select the trigger line used for trigger signal generation.	
DioCl1PwmOut_setTriggerLineOutStatus	336
To set the status of the trigger line.	
DioCl1PwmOut_setTriggerMode	338
To set the trigger mode of a DIO Class 1 PWM output channel.	
DioCl1PwmOut_setUpdateMode	339
To set the update mode of the DIO Class 1 PWM output channel.	
DioCl1PwmOut_start	340
To start the PWM signal generation on a DIO Class 1 PWM output channel.	
DioCl1PwmOut_write	341
To update the data of the specified DIO Class 1 PWM output channel during run time.	

Information in other sections

Multichannel PWM Signal Generation.....	771
Block-Commutated PWM Generation.....	725

DioCl1PwmOut_apply

Syntax

```
UInt32 DioCl1PwmOut_apply(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To apply the initialization settings to the DIO Class 1 PWM output channels.

Description

Before you can start the specified DIO Class 1 channel, you have to transfer the settings of the DioCl1PwmOut driver object to its related output channel by using `DioCl1PwmOut_apply`. You have to do this also for the default values and for values that you specified via the `DioCl1PwmOut_set<Parameter>` functions. The `DioCl1PwmOut_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The signal generation on the specified digital output channel is not activated until you have called `DioCl1PwmOut_start`.

To update settings during run time, you have to use `DioCl1PwmOut_write`.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using `DioCl1PwmOut_create`.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PwmOut_create

Syntax	<pre>UInt32 DioCl1PwmOut_create(DioCl1PwmOutSDrvObject **ppDioCl1PwmOut, UInt32 Port, UInt32 Channel)</pre>
Include file	<code>IoDrvDioClass1PwmOut.h</code>
Purpose	To create the I/O driver object for PWM signal generation using a DIO Class 1 output channel.
Description	<p>This function performs all the steps necessary to create an I/O driver object for the DIO Class 1 unit. A DioCl1PwmOut driver object handles one digital output channel for PWM signal generation.</p> <p>The specified DIO Class 1 channel is initialized with default values so that it is ready for use.</p> <p>The initial values are:</p> <ul style="list-style-type: none"> ▪ Output signal voltage level: 2.5 V ▪ PWM update mode: synchronous update ▪ PWM signal period: 1 ms ▪ PWM signal duty cycle: 50% ▪ Inverting mode: do not invert output signal ▪ Rising edge delay: 0 s ▪ Interrupt mode: no interrupt generation ▪ Trigger mode: no trigger signal generation ▪ Event downsample: 1 ▪ Event delay: 0 s <p>The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

ppDioCl1PwmOut Lets you specify the address of a variable which holds the address of the created driver object.

Port Lets you specify the number of the DIO Class 1 port providing the required channels in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
DIO_CLASS1_PORT_2	Specifies port 2 of the DIO Class 1 unit.
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

[DioCl1PwmOut_disableInterrupts](#)**Syntax**

```
UInt32 DioCl1PwmOut_disableInterrupts(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    UInt32 InterruptSelect)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To disable the generation of interrupts on a DIO Class 1 PWM output channel.

Description	<p>With this function, you can disable the generation of interrupts that you specified before by using <code>DioCl1PwmOut_setInterruptMode</code>. The settings take effect immediately.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using <code>DioCl1PwmOut_create</code>.</p> <p>InterruptSelect Lets you select the interrupts that are to be disabled. The symbols can be combined by using the bitwise OR operator.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_INT_RISING_EDGE</td><td>Disables the generation of interrupts on rising edges.</td></tr> <tr> <td>DIO_CLASS1_INT_FALLING_EDGE</td><td>Disables the generation of interrupts on falling edges.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_INT_RISING_EDGE	Disables the generation of interrupts on rising edges.	DIO_CLASS1_INT_FALLING_EDGE	Disables the generation of interrupts on falling edges.
Symbol	Meaning						
DIO_CLASS1_INT_RISING_EDGE	Disables the generation of interrupts on rising edges.						
DIO_CLASS1_INT_FALLING_EDGE	Disables the generation of interrupts on falling edges.						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PwmOut_enableInterrupts

Syntax	<pre>UInt32 DioCl1PwmOut_enableInterrupts(DioCl1PwmOutSDrvObject *pDioCl1PwmOut, UInt32 InterruptSelect)</pre>
Include file	<code>IoDrvDioClass1PwmOut.h</code>

Purpose	To enable the generation of interrupts on a DIO Class 1 PWM output channel.						
Description	<p>With this function, you can either enable the generation of interrupts or reenable the generation of interrupts that you disabled before by using <code>DioCl1PwmOut_disableInterrupts</code>. The interrupts have to be specified before by using <code>DioCl1PwmOut_setInterruptMode</code>. The settings take effect immediately.</p> <p>After calling <code>DioCl1PwmOut_start</code>, interrupts are disabled.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using <code>DioCl1PwmOut_create</code>.</p> <p>InterruptSelect Lets you select the interrupts that are to be enabled. The symbols can be combined by using the bitwise OR operator.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_INT_RISING_EDGE</td><td>Enables the generation of interrupts on rising edges.</td></tr> <tr> <td>DIO_CLASS1_INT_FALLING_EDGE</td><td>Enables the generation of interrupts on falling edges.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_INT_RISING_EDGE	Enables the generation of interrupts on rising edges.	DIO_CLASS1_INT_FALLING_EDGE	Enables the generation of interrupts on falling edges.
Symbol	Meaning						
DIO_CLASS1_INT_RISING_EDGE	Enables the generation of interrupts on rising edges.						
DIO_CLASS1_INT_FALLING_EDGE	Enables the generation of interrupts on falling edges.						
Return value	The function returns an error code.						
Related topics	<p>References</p> <table border="1"> <tr> <td><code>DioCl1PwmOut_disableInterrupts.....</code></td><td>321</td></tr> <tr> <td><code>DioCl1PwmOut_setInterruptMode.....</code></td><td>327</td></tr> </table>	<code>DioCl1PwmOut_disableInterrupts.....</code>	321	<code>DioCl1PwmOut_setInterruptMode.....</code>	327		
<code>DioCl1PwmOut_disableInterrupts.....</code>	321						
<code>DioCl1PwmOut_setInterruptMode.....</code>	327						

DioCl1PwmOut_setDutyCycle

Syntax

```
UInt32 DioCl1PwmOut_setDutyCycle(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    Float64 PwmDutyCycle)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To set the duty cycle of the PWM signal to be output using a DIO Class 1 channel.

Description

The duty cycle is defined as the ratio between T_{High} and T_{Period} . You can specify an initial value for the duty cycle or you can update the duty cycle during run time of the real-time application.

The resolution of the duty cycle depends on the specified period value.

The resolution for time intervals within the PWM signal generator unit is 10 ns.

The setting does not take effect until you call `DioCl1PwmOut_apply` during the initialization phase or `DioCl1PwmOut_write` during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the PWM output that you specified by using `DioCl1PwmOut_create`, you must use `DioCl1PwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using `DioCl1PwmOut_create`.

PwmDutyCycle Lets you specify the duty cycle of the PWM signal to be generated in the range 0.0 ... 1.0 (0 ... 100%).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setEventDelay

Syntax

```
UInt32 DioCl1PwmOut_setEventDelay(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    Float64 EventDelay)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To set the delay for events on a DIO Class 1 output channel.

Description

The function is used to delay the generation of events that you specified by using `DioCl1PwmOut_setInterruptMode` and `DioCl1PwmOut_setTriggerMode`. With these functions, you also specified the channel that is affected by the delay setting.

The delay is the time interval between the occurrence of the event and the generation of the event signal (interrupt or trigger signal).

The setting does not take effect until you have called `DioCl1PwmOut_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using `DioCl1PwmOut_create`.

EventDelay Lets you specify the time interval by which the generation of an event is delayed in seconds in the range 0 ... 1.34 s. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setEventDownsample

Syntax

```
UInt32 DioCl1PwmOut_setEventDownsample(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    UInt32 DownsamplingValue)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To set the downsampling for events on a DIO Class 1 output channel.

Description

The function is used to reduce the number of generated events that you specified by using **DioCl1DigIn_setInterruptMode** and **DioCl1DigIn_setTriggerMode**. With these functions, you also specified the channel that is affected by the downsampling setting.

The setting does not take effect until you have called **DioCl1PwmOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using **DioCl1PwmOut_create**.

DownsamplingValue Lets you specify the downsampling value for event generation (interrupt or trigger signal) in the range 1 ... 256.

The downsampling value specifies whether to generate an event on each occurrence of the event condition (DownsamplingValue=1, default) or only on each n-th occurrence, where n is the specified downsampling value.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setInterruptMode

Syntax

```
UInt32 DioCl1PwmOut_setInterruptMode(
    DioCl1PPwmOutSDrvObject *pDioCl1PwmOut,
    UInt32 InterruptMode)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To set the interrupt mode of a DIO Class 1 output channel.

Description

This function is used to generate interrupts on rising, falling, or both edges of the output signal connected to the specified channel. The detection of the edge type is performed after an optional signal inverting. You can configure the inverting mode by using **DioCl1PwmOut_setInvertingMode**.

If you want to configure a combination of interrupts and trigger signals, you have to configure the trigger mode in addition to the interrupt mode by using **DioCl1PwmOut_setTriggerMode**.

The setting does not take effect until you have called **DioCl1PwmOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using **DioCl1PwmOut_create**.

InterruptMode Lets you specify the interrupt mode. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_NO_INTERRUPT	No interrupt is to be generated (default).
DIO_CLASS1_INT_RISING_EDGE	Interrupts are generated on rising edges.
DIO_CLASS1_INT_FALLING_EDGE	Interrupts are generated on falling edges.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setInvertingMode

Syntax

```
UInt32 DioCl1PwmOut_setInvertingMode(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    UInt32 InvertingMode)
```

Include file

IoDrvDioClass1PwmOut.h

Purpose

To set the inverting mode of the DIO Class 1 PWM output channel.

Description

With this function, you can specify whether the PWM output signal is to be inverted at the output channel.

The setting does not take effect until you have called **DioCl1PwmOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using [DioCl1PwmOut_create](#).

InvertingMode Lets you select the inverting mode used when generating an output signal.

Symbol	Meaning
DIO_CLASS1_NOT_INVERTED	The output signal is not to be inverted (default).
DIO_CLASS1_INVERTED	The output signal is to be inverted.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setIoIntVector

Syntax

```
UInt32 DioCl1PwmOut_setIoIntVector(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    UInt32 InterruptSelect,
    IolibTIOIntHandler InterruptHandler)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To set the interrupt handler for processing interrupts from a DIO Class 1 PWM output channel.

Description

This function is used to register the function that handles the generated interrupts specified by using [DioCl1PwmOut_setInterruptMode](#). For each specified condition for interrupt generation (rising edge, falling edge), you can define separate functions. If you want to handle all conditions with the same function, you can use the bitwise OR operator to combine the predefined symbols of the **InterruptSelect** parameter.

Note

The condition for interrupt generation must be the same as specified for the **DioCl1PwmOut_setInterruptMode** function.

The setting does not take effect until you have called **DioCl1PwmOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using **DioCl1PwmOut_create**.

InterruptSelect Lets you specify the interrupts the given interrupt handler is to process. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_RISING_EDGE	The handler is to process interrupts generated on rising edges.
DIO_CLASS1_INT_FALLING_EDGE	The handler is to process interrupts generated on falling edges.

InterruptHandler Lets you specify the address of the function which is to handle the generated interrupt.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setOutputHighZ

Syntax

```
UInt32 DioCl1PwmOut_setOutputHighZ(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    UInt32 HighZStateOnOff)
```

Include file	<code>IoDrvDioClass1PwmOut.h</code>						
Purpose	To set the high impedance state of the DIO Class 1 PWM output channel.						
Description	<p>You can force the output channel of the driver object to the high impedance state, for example, for termination purposes, by setting the high impedance state to <i>On</i>. If you want the real-time application to control the output levels, you can release the high impedance state again.</p> <p>The setting takes effect only after you call <code>DioCl1PwmOut_start</code> or <code>DioCl1PwmOut_write</code> during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using <code>DioCl1PwmOut_create</code>.</p> <p>HighZStateOnOff Lets you specify if outputs are to be set to the high impedance state or if outputs are to be released from the high impedance state.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>DIO_CLASS1_HIGH_Z_OFF</code></td><td>Releases the high impedance state.</td></tr> <tr> <td><code>DIO_CLASS1_HIGH_Z_ON</code></td><td>Sets outputs to the high impedance state (default).</td></tr> </tbody> </table>	Symbol	Meaning	<code>DIO_CLASS1_HIGH_Z_OFF</code>	Releases the high impedance state.	<code>DIO_CLASS1_HIGH_Z_ON</code>	Sets outputs to the high impedance state (default).
Symbol	Meaning						
<code>DIO_CLASS1_HIGH_Z_OFF</code>	Releases the high impedance state.						
<code>DIO_CLASS1_HIGH_Z_ON</code>	Sets outputs to the high impedance state (default).						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						

DioCl1PwmOut_setPeriod

Syntax

```
UInt32 DioCl1PwmOut_setPeriod(  
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,  
    Float64 PwmPeriod)
```

Include file

IoDrvDioClass1PwmOut.h

Purpose

To set the period of the PWM signal to be output at a DIO Class 1 channel.

Description

The specified value for the PWM period is stored in an internal buffer and not immediately updated at the output channels.

The setting does not take effect until you call **DioCl1PwmOut_apply** during the initialization phase or **DioCl1PwmOut_write** during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the PWM output that you specified by using **DioCl1PwmOut_create**, you must use **DioCl1PwmOut_start**.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using **DioCl1PwmOut_create**.

PwmPeriod Lets you specify the period of the PWM signal to be generated in seconds in the range 100 ns ... 1.34 s. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setRisingEdgeDelay

Syntax

```
UInt32 DioCl1PwmOut_setRisingEdgeDelay(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    Float64 RisingEdgeDelay)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To set the time for delaying the rising edges of the DIO Class 1 PWM output channel.

Description

The delay will be applied to the channel that is controlled by the specified DioCl1PwmOut driver object.

The generated rising edges of the output signal will be output after the specified time. The detection of the rising edges is performed after an optional signal inverting. You can configure the inverting mode by using [DioCl1PwmOut_setInvertingMode](#).

The setting does not take effect until you have called [DioCl1PwmOut_apply](#) during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using [DioCl1PwmOut_create](#).

RisingEdgeDelay Lets you specify the delay applied to the rising edges of the output signal in seconds in the range 0 ... 655 µs. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setSignalVoltage

Syntax

```
UInt32 DioCl1PwmOut_setSignalVoltage(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    UInt32 SignalVoltageSelector)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To set the voltage level used for the DIO Class 1 PWM output channel.

Description

The output voltage level will be applied to the PWM output channel that is controlled by the specified DioCl1PwmOut driver object.

The setting does not take effect until you have called **DioCl1PwmOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using **DioCl1PwmOut_create**.

SignalVoltageSelector Lets you select the voltage level used for the output signal(s).

Symbol	Meaning
DIO_CLASS1_SIGNAL_2_5_V	Specifies 2.5 V as the output voltage level (default).
DIO_CLASS1_SIGNAL_3_3_V	Specifies 3.3 V as the output voltage level.
DIO_CLASS1_SIGNAL_5_0_V	Specifies 5.0 V as the output voltage level.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setTriggerLineOut

Syntax

```
UInt32 DioCl1PwmOut_setTriggerLineOut(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    UInt32 TriggerLineOut)
```

Include file

IoDrvDioClass1PwmOut.h

Purpose

To select the trigger line used for trigger signal generation.

Description

A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board. You have to specify the trigger line the consumer must listen to by using the relevant `xxx_setTriggerLineIn` function. A trigger line can be allocated only once.

Before you can use trigger signals, you have to set the trigger mode by using `DioCl1PwmOut_setTriggerMode`. With `DioCl1PwmOut_setTriggerLineOutStatus` you enable or disable the trigger line.

The setting does not take effect until you have called `DioCl1PwmOut_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) .								
	For more information on the I/O mapping, refer to PWM Signal Generation (DIO Class 1) (MicroLabBox Features) .								
Parameters	<p>pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using DioCl1PwmOut_create.</p> <p>TriggerLineOut Lets you specify the number of the trigger line to be used for transmitting the trigger signal in the range 1 ... 16.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_TRIGGER_LINE_1</td><td>Specifies trigger line 1.</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td>DIO_CLASS1_TRIGGER_LINE_16</td><td>Specifies trigger line 16.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.	DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.
Symbol	Meaning								
DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.								
...	...								
DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.								
Return value	The function returns an error code.								

Related topics	References
DioCl1PwmOut_setTriggerLineOutStatus 336	

DioCl1PwmOut_setTriggerLineOutStatus

Syntax	<pre>UInt32 DioCl1PwmOut_setTriggerLineOutStatus(DioCl1PwmOutSDrvObject *pDioCl1PwmOut, UInt32 TriggerLineStatus)</pre>
Include file	<code>IoDrvDioClass1PwmOut.h</code>
Purpose	To set the status of the trigger line.

Description With this function, you can enable or disable the trigger line that you specified before by using [DioCl1PwmOut_setTriggerLineOut](#).

The function is intended to be called during the initialization phase of the real-time application or during a run-to-stop or a stop-to-run transition of the real-time application.

The setting does not take effect until you call [DioCl1PwmOut_start](#).

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters **pDioCl1PwmOut** Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using [DioCl1PwmOut_create](#).

TriggerLineStatus Lets you specify the status of the trigger line.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_DISABLE	Disables the trigger line.
DIO_CLASS1_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1PwmOut_create.....	320
DioCl1PwmOut_start.....	340

DioCl1PwmOut_setTriggerMode

Syntax	<pre>UInt32 DioCl1PwmOut_setTriggerMode(DioCl1PwmOutSDrvObject *pDioCl1PwmOut, UInt32 TriggerMode)</pre>
Include file	<code>IoDrvDioClass1PwmOut.h</code>
Purpose	To set the trigger mode of a DIO Class 1 PWM output channel.
Description	<p>This function is used to generate trigger signals on rising, falling, or both edges of the output signal connected to the specified channel. The detection of the edge type is performed after an optional signal inverting. You can configure the inverting mode by using DioCl1PwmOut_setInvertingMode. Additionally, you have to specify the trigger line for the generated trigger signal by using DioCl1PwmOut_setTriggerLineOut. The I/O unit that is to consume the trigger signal has to specify this trigger line by using its related xxx_setTriggerLineIn function.</p> <p>If you want to configure a combination of interrupts and trigger signals, you have to configure the interrupt mode in addition to the trigger mode by using DioCl1PwmOut_setInterruptMode.</p> <p>The setting does not take effect until you have called DioCl1PwmOut_apply during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>
Parameters	pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using DioCl1PwmOut_create .

TriggerMode Lets you specify the trigger mode. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_NO_TRIGGER	No trigger is to be generated (default).
DIO_CLASS1_TRIG_RISING_EDGE	Triggers are generated on rising edges.
DIO_CLASS1_TRIG_FALLING_EDGE	Triggers are generated on falling edges.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmOut_setUpdateMode

Syntax

```
UInt32 DioCl1PwmOut_setUpdateMode(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut,
    UInt32 UpdateMode)
```

Include file

`IoDrvDioClass1PwmOut.h`

Purpose

To set the update mode of the DIO Class 1 PWM output channel.

Description

If a parameter of the PWM output signal has to be changed, you can specify whether the update will be synchronized to the period.

- Synchronous update mode
No modification of a parameter of the PWM output signal is applied until the beginning of the next period.
- Asynchronous update mode
Any modification of a parameter of the PWM output signal is applied immediately (when calling `DioCl1PwmOut_write`).

For further information on the update mode, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

The setting does not take effect until you have called `DioCl1PwmOut_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) .						
	For more information on the I/O mapping, refer to PWM Signal Generation (DIO Class 1) (MicroLabBox Features) .						
Parameters	<p>pDioCl1PwmOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using DioCl1PwmOut_create.</p> <p>UpdateMode Lets you specify the update mode used when generating a PWM output signal.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_PWM_ASYNC_UPDATE</td><td>Selects the asynchronous update mode.</td></tr> <tr> <td>DIO_PWM_SYNC_UPDATE</td><td>Selects the synchronous update mode (default).</td></tr> </tbody> </table>	Symbol	Meaning	DIO_PWM_ASYNC_UPDATE	Selects the asynchronous update mode.	DIO_PWM_SYNC_UPDATE	Selects the synchronous update mode (default).
Symbol	Meaning						
DIO_PWM_ASYNC_UPDATE	Selects the asynchronous update mode.						
DIO_PWM_SYNC_UPDATE	Selects the synchronous update mode (default).						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PwmOut_start

Syntax	<pre>UInt32 DioCl1PwmOut_start(DioCl1PwmOutSDrvObject *pDioCl1PwmOut)</pre>
Include file	<code>IoDrvDioClass1PwmOut.h</code>
Purpose	To start the PWM signal generation on a DIO Class 1 PWM output channel.
Description	This function starts the specified DioCl1PwmOut driver so that it will activate its digital output channel. The digital output channel is enabled to generate the output signal. If you call DioCl1PwmOut_setOutputHighZ(DIO_CLASS1_HIGH_Z_ON) before, the output channel stays in the high impedance state.

`DioCl1PwmOut_start` also enables the configured trigger line. If you call `DioCl1PwmOut_setTriggerLineOutStatus(DIO_CLASS1_TRIGGER_LINE_D_ISABLE)` before, the trigger line stays disabled.

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the PWM output that you specified by using `DioCl1PwmOut_create`, you must use `DioCl1PwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

`pDioCl1PwmOut` Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using `DioCl1PwmOut_create`.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics

References

<code>DioCl1PwmOut_setOutputHighZ.....</code>	330
<code>DioCl1PwmOut_setTriggerLineOutStatus.....</code>	336

DioCl1PwmOut_write

Syntax

```
UInt32 DioCl1PwmOut_write(
    DioCl1PwmOutSDrvObject *pDioCl1PwmOut)
```

Include file	<code>IoDrvDioClass1PwmOut.h</code>						
Purpose	To update the data of the specified DIO Class 1 PWM output channel during run time.						
Description	<p>If have used one of the following functions during run time, their settings will not be updated on the hardware until you have called <code>DioCl1PwmOut_write</code>.</p> <ul style="list-style-type: none"> ▪ <code>DioCl1PwmOut_setPeriod</code> ▪ <code>DioCl1PwmOut_setDutyCycle</code> ▪ <code>DioCl1PwmOut_setOutputHighZ</code> <p>The specified digital output channel must be enabled by using <code>DioCl1PwmOut_start</code>.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p><code>pDioCl1PwmOut</code> Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmOut driver object had to be created before by using <code>DioCl1PwmOut_create</code>.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						

PWM Signal Measurement (PWM In)

Introduction	MicroLabBox provides functions to measure PWM signals.
	For details, refer to PWM Signal Measurement (DIO Class 1) (MicroLabBox Features) .

Where to go from here

Information in this section

DioCl1Pwmln_apply	344
To apply the initialization settings to the DIO Class 1 PWM input channel.	
DioCl1Pwmln_create	345
To create the I/O driver object for PWM signal measurement via DIO Class 1 input channel.	
DioCl1Pwmln_disableInterrupts	346
To disable interrupt generation on a DIO Class 1 PWM input channel.	
DioCl1Pwmln_enableInterrupts	347
To enable interrupt generation on a DIO Class 1 PWM input channel.	
DioCl1Pwmln_getDutyCycle	349
To get the duty cycle of the PWM signal captured by a DIO Class 1 PWM input channel.	
DioCl1Pwmln_getFrequency	350
To get the frequency of the PWM signal captured by a DIO Class 1 PWM input channel.	
DioCl1Pwmln_read	351
To read the data from a DIO Class 1 PWM input channel.	
DioCl1Pwmln_setEventDelay	352
To set the delay value for events generated by a DIO Class 1 PWM input channel.	
DioCl1Pwmln_setEventDownsample	353
To set the downsampling value for events generated by a DIO Class 1 PWM input channel.	
DioCl1Pwmln_setInterruptMode	354
To set the interrupt mode of a DIO Class 1 PWM input channel.	
DioCl1Pwmln_setIntVect	356
To set the interrupt handler for processing interrupts from a DIO Class 1 PWM input channel.	
DioCl1Pwmln_setTimeout	357
To set the timeout value for signal measurement via a DIO Class 1 PWM input channel.	
DioCl1Pwmln_setTriggerMode	358
To set the trigger mode of a DIO Class 1 PWM input channel.	
DioCl1Pwmln_setTriggerLineOut	360
To select the trigger line used for trigger signal on specific condition on DIO Class 1 PWM input channel.	
DioCl1Pwmln_setTriggerLineOutStatus	361
To set the status of the trigger line.	
DioCl1Pwmln_setUpdateMode	362
To set the update mode of the DIO Class 1 PWM input channel.	

DioCl1PwmIn_start..... 363
To start the DIO Class 1 PWM input channel.

DioCl1PwmIn_apply

Syntax `UInt32 DioCl1PwmIn_apply(DioCl1PwmInSDrvObject *pDioCl1PwmIn)`

Include file `IoDrvDioClass1PwmIn.h`

Purpose To apply the initialization settings to the DIO Class 1 PWM input channel.

Description Before you can start the specified input channel of the DIO Class 1 unit, you have to transfer the settings of the DioCl1PwmIn driver object to its related input channel by using `DioCl1PwmIn_apply`. You have to do this also for the default values and for values that you specified by using the `DioCl1PwmIn_set<Parameter>` functions. The `DioCl1PwmIn_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The signal measurement on the specified digital input channel is not activated until you have called `DioCl1PwmIn_start`.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters `pDioCl1PwmIn` Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using `DioCl1PwmIn_create`.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmIn_create

Syntax

```
UInt32 DioCl1PwmIn_create(
    DioCl1PwmInSDrvObject **ppDioCl1PwmIn,
    UInt32 Port,
    UInt32 Channel)
```

Include file

`IoDrvDioClass1PwmIn.h`

Purpose

To create the I/O driver object for PWM signal measurement via DIO Class 1 input channel.

Description

This function performs all the steps necessary to create an I/O driver object for the DIO Class 1 unit. A `DioCl1PwmIn` driver object handles one digital input channel for PWM signal measurement.

The specified DIO Class 1 channel is initialized with default values so that it will be ready for use.

The initial values are:

- PWM input update mode: rising edge
- Edge polarity: rising edge
- Interrupt mode: no interrupt generation
- Trigger mode: no trigger signal generation
- Event downsample: 1
- Event delay: 0 s
- Timeout: 1.34 s

The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters **ppDioCl1PwmIn** Lets you specify the address of a variable which holds the address of the created driver object.

Port Lets you specify the number of the DIO Class 1 port providing the required channels in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
DIO_CLASS1_PORT_2	Specifies port 2 of the DIO Class 1 unit.
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmIn_disableInterrupts

Syntax

```
UInt32 DioCl1PwmIn_disableInterrupts(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    UInt32 InterruptSelect)
```

Include file

IoDrvDioClass1PwmIn.h

Purpose

To disable generation of interrupts on a DIO Class 1 PWM input channel.

Description	<p>With this function, you can disable the generation of interrupts that you specified before by using DioCl1PwmIn_setInterruptMode. The settings take effect immediately.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Measurement (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using DioCl1PwmIn_create.</p> <p>InterruptSelect Lets you select the interrupts that are to be disabled. The symbols can be combined by using the bitwise OR operator.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_INT_RISING_EDGE</td><td>Disables the generation of interrupts on rising edges.</td></tr> <tr> <td>DIO_CLASS1_INT_FALLING_EDGE</td><td>Disables the generation of interrupts on falling edges.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_INT_RISING_EDGE	Disables the generation of interrupts on rising edges.	DIO_CLASS1_INT_FALLING_EDGE	Disables the generation of interrupts on falling edges.
Symbol	Meaning						
DIO_CLASS1_INT_RISING_EDGE	Disables the generation of interrupts on rising edges.						
DIO_CLASS1_INT_FALLING_EDGE	Disables the generation of interrupts on falling edges.						
Return value	The function returns an error code.						

DioCl1PwmIn_enableInterrupts

Syntax	<pre>UInt32 DioCl1PwmIn_enableInterrupts(DioCl1PwmInSDrvObject *pDioCl1PwmIn, UInt32 InterruptSelect)</pre>
Include file	<code>IoDrvDioClass1PwmIn.h</code>

Purpose	To enable generation of interrupts on a DIO Class 1 PWM input channel.						
Description	<p>With this function, you can either enable the generation of interrupts or reenable the generation of interrupts that you disabled before by using <code>DioCl1PwmIn_disableInterrupts</code>. The interrupts have to be specified before by using <code>DioCl1PwmIn_setInterruptMode</code>. The settings take effect immediately.</p> <p>After calling <code>DioCl1PwmIn_start</code>, interrupts are disabled.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Measurement (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p><code>pDioCl1PwmIn</code> Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using <code>DioCl1PwmIn_create</code>.</p> <p><code>InterruptSelect</code> Lets you select the interrupts that are to be enabled. The symbols can be combined by using the bitwise OR operator.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>DIO_CLASS1_INT_RISING_EDGE</code></td><td>Enables the generation of interrupts on rising edges.</td></tr> <tr> <td><code>DIO_CLASS1_INT_FALLING_EDGE</code></td><td>Enables the generation of interrupts on falling edges.</td></tr> </tbody> </table>	Symbol	Meaning	<code>DIO_CLASS1_INT_RISING_EDGE</code>	Enables the generation of interrupts on rising edges.	<code>DIO_CLASS1_INT_FALLING_EDGE</code>	Enables the generation of interrupts on falling edges.
Symbol	Meaning						
<code>DIO_CLASS1_INT_RISING_EDGE</code>	Enables the generation of interrupts on rising edges.						
<code>DIO_CLASS1_INT_FALLING_EDGE</code>	Enables the generation of interrupts on falling edges.						
Return value	The function returns an error code.						
Related topics	<p>References</p> <table border="1"> <tr> <td><code>DioCl1PwmIn_disableInterrupts</code>.....</td><td>346</td></tr> <tr> <td><code>DioCl1PwmIn_setInterruptMode</code>.....</td><td>354</td></tr> </table>	<code>DioCl1PwmIn_disableInterrupts</code>	346	<code>DioCl1PwmIn_setInterruptMode</code>	354		
<code>DioCl1PwmIn_disableInterrupts</code>	346						
<code>DioCl1PwmIn_setInterruptMode</code>	354						

DioCl1PwmIn_getDutyCycle

Syntax

```
UInt32 DioCl1PwmIn_getDutyCycle(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    Float64 *pPwmDutyCycle)
```

Include file

`IoDrvDioClass1PwmIn.h`

Purpose

To get the duty cycle of the PWM signal captured by a DIO Class 1 PWM input channel.

Description

Because you get the values from the internal buffer of the driver object, there might be newer input values. To update the internal buffer with the latest values, you have to call `DioCl1PwmIn_read` beforehand.

The measured duty cycle can be in the range 0.0 ... 1.0, representing a duty cycle of 0 ... 100%. The PWM signal measurement supports measuring input signals with a frequency of 0.5 Hz up to 10 MHz with a resolution of the period of 10 ns. However, these limits are true only if the input PWM signals have a duty cycle of 50%.

The width of the low level portion or the high level portion of the period must be at least 50 ns and at most 1 s.

If the maximum input frequency is exceeded or the duty cycle near the maximum frequency is different from 50%, the resulting value of the duty cycle cannot be predicted. If the frequency of the PWM input signal goes below the minimum value, the function will output 0.0 or 1.0 as the duty cycle.

- Duty cycle = 0.0
 - An overflow is detected in the low level portion of the PWM period.
- Duty cycle = 1.0
 - An overflow is detected in the high level portion of the PWM period.

Note

All digital inputs of the DIO Class 1 unit are in a disabled state after reset and will read 0. To enable the PWM input channel that you specified by using `DioCl1PwmIn_create`, you must use `DioCl1PwmIn_start`.

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using [DioCl1PwmIn_create](#).

pPwmDutyCycle Lets you specify the address of a variable that holds the measured duty cycle in the range 0.0 ... 1.0 (0 ... 100%).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmIn_getFrequency

Syntax

```
UInt32 DioCl1PwmIn_getFrequency(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    Float64 *pPwmFrequency)
```

Include file

IoDrvDioClass1PwmIn.h

Purpose

To get the frequency of the PWM signal captured by a DIO Class 1 PWM input channel.

Description

Because you get the values from the internal buffer of the driver object, there might be newer input values. To update the internal buffer with the latest values, you have to call [DioCl1PwmIn_read](#) beforehand.

The PWM signal measurement supports measuring input signals with a frequency of 0.5 Hz up to 10 MHz with a resolution of the period of 10 ns. However, these limits are true only if the input PWM signals have a duty cycle of 50%.

If the maximum input frequency is exceeded or the duty cycle near the maximum frequency is different from 50%, the resulting value of the frequency cannot be predicted. If the frequency of the PWM input signal goes below the minimum value, the function will output 0.0 Hz as the result. If the function delivers

0.0 Hz, the function `DioCl1PwmIn_getDutyCycle` will output 0.0 or 1.0 as the corresponding duty cycle of the PWM input signal.

Note

All digital inputs of the DIO Class 1 unit are in a disabled state after reset and will read 0. To enable the PWM input channel that you specified by using `DioCl1PwmIn_create`, you must use `DioCl1PwmIn_start`.

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The `DioCl1PwmIn` driver object had to be created before by using `DioCl1PwmIn_create`.

pPwmFrequency Lets you specify the address of a variable that holds the measured frequency in Hz. Valid values are in the range 0.5 Hz ... 10 MHz.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

DioCl1PwmIn_read

Syntax

```
UInt32 DioCl1PwmIn_read(DioCl1PwmInSDrvObject *pDioCl1PwmIn)
```

Include file

`IoDrvDioClass1PwmIn.h`

Purpose

To read the data from a DIO Class 1 PWM input channel.

Description	<p>This function reads the raw data of the PWM input signal captured by the specified PWM input channel. It calculates the frequency and the duty cycle of the PWM signal and stores the results as a consistent data set in an internal buffer.</p> <p>The input channel must be enabled before by using DioCl1PwmIn_start.</p> <p>To get the measured values from the internal buffer, you must use one of the following functions.</p> <ul style="list-style-type: none"> ▪ DioCl1PwmIn_getFrequency to get the measured value for the frequency. ▪ DioCl1PwmIn_getDutyCycle to get the measured value for the duty cycle. <p>The function is intended to be used during run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Measurement (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using DioCl1PwmIn_create.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PwmIn_setEventDelay

Syntax	<pre>UInt32 DioCl1PwmIn_setEventDelay (DioCl1PwmInSDrvObject *pDioCl1PwmIn, Float64 EventDelay)</pre>
Include file	<code>IoDrvDioClass1PwmIn.h</code>
Purpose	To set the delay value for events generated by a DIO Class 1 PWM input channel.

Description	<p>The function is used to delay the generation of events that you specified by using DioCl1PwmIn_SetInterruptMode and DioCl1PwmIn_SetTriggerMode. With these functions, you also specified the channel that is affected by the delay setting.</p> <p>The delay is the time interval between the occurrence of the event and the generation of the event signal (interrupt or trigger signal).</p> <p>The setting does not take effect until you have called DioCl1PwmIn_Apply during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Measurement (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using DioCl1PwmIn_Create.</p> <p>EventDelay Lets you specify the time interval by which the generation of an event is delayed in seconds in the range 0 ... 1.34 s. You can specify the value in steps of 10 ns.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PwmIn_SetEventDownsample

Syntax	<pre>UInt32 DioCl1PwmIn_SetEventDownsample(DioCl1PwmInSDrvObject *pDioCl1PwmIn, UInt32 DownsamplingValue)</pre>
---------------	--

Include file	<code>IoDrvDioClass1PwmIn.h</code>
---------------------	------------------------------------

Purpose	To set the downsampling value for events generated by a DIO Class 1 PWM input channel.						
Description	<p>The function is used to reduce the number of generated events that you specified by using <code>DioCl1PwmIn_setInterruptMode</code> and <code>DioCl1PwmIn_setTriggerMode</code>. With these functions, you also specified the channel that is affected by the downsampling setting.</p> <p>The setting does not take effect until you have called <code>DioCl1PwmIn_apply</code> during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Measurement (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using <code>DioCl1PwmIn_create</code>.</p> <p>DownsamplingValue Lets you specify the downsampling value for event generation (interrupt or trigger signal) in the range 1 ... 256. The downsampling value specifies whether to generate an event on each occurrence of the event condition (DownsamplingValue=1, default) or only on each n-th occurrence, where n is the specified downsampling value.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PwmIn_setInterruptMode

Syntax

```
UInt32 DioCl1PwmIn_setInterruptMode(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    UInt32 InterruptMode)
```

Include file	<code>IoDrvDioClass1PwmIn.h</code>								
Purpose	To set the interrupt mode of a DIO Class 1 PWM input channel.								
Description	<p>This function is used to generate interrupts on rising, falling, or both edges of the input signal connected to the specified channel.</p> <p>If you want to configure a combination of interrupts and trigger signals, you have to configure the trigger mode in addition to the interrupt mode by using <code>DioCl1PwmIn_setTriggerMode</code>.</p> <p>The setting does not take effect until you have called <code>DioCl1PwmIn_apply</code> during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>								
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Measurement (DIO Class 1) (MicroLabBox Features).</p>								
Parameters	<p>pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using <code>DioCl1PwmIn_create</code>.</p> <p>InterruptMode Lets you specify the interrupt mode. The symbols can be combined by using the bitwise OR operator.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>DIO_CLASS1_NO_INTERRUPT</code></td><td>No interrupt is to be generated (default).</td></tr> <tr> <td><code>DIO_CLASS1_INT_RISING_EDGE</code></td><td>Interrupts are generated on rising edges.</td></tr> <tr> <td><code>DIO_CLASS1_INT_FALLING_EDGE</code></td><td>Interrupts are generated on falling edges.</td></tr> </tbody> </table>	Symbol	Meaning	<code>DIO_CLASS1_NO_INTERRUPT</code>	No interrupt is to be generated (default).	<code>DIO_CLASS1_INT_RISING_EDGE</code>	Interrupts are generated on rising edges.	<code>DIO_CLASS1_INT_FALLING_EDGE</code>	Interrupts are generated on falling edges.
Symbol	Meaning								
<code>DIO_CLASS1_NO_INTERRUPT</code>	No interrupt is to be generated (default).								
<code>DIO_CLASS1_INT_RISING_EDGE</code>	Interrupts are generated on rising edges.								
<code>DIO_CLASS1_INT_FALLING_EDGE</code>	Interrupts are generated on falling edges.								
Return value	The function returns an error code.								
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.		
Error Code	Meaning								
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.								
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.								

DioCl1PwmIn_SetIoIntVector

Syntax

```
UInt32 DioCl1PwmIn_SetIoIntVector(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    UInt32 InterruptSelect,
    IoLibTIOIntHandler InterruptHandler)
```

Include file

`IoDrvDioClass1PwmIn.h`

Purpose

To set the interrupt handler for processing interrupts from a DIO Class 1 PWM input channel.

Description

This function is used to register the function that handles the generated interrupts specified by using `DioCl1PwmIn_SetInterruptMode`. For each specified condition for interrupt generation (rising edge, falling edge), you can define separate functions. If you want to handle all the conditions with the same function, you can use the bitwise OR operator to combine the predefined symbols of the `InterruptSelect` parameter.

Note

The condition for interrupt generation must be the same as specified for the `DioCl1PwmIn_SetInterruptMode` function.

The setting does not take effect until you have called `DioCl1PwmIn_Apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

`pDioCl1PwmIn` Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using `DioCl1PwmIn_Create`.

InterruptSelect Lets you specify the interrupts the given interrupt handler is to process. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_RISING_EDGE	The handler is to process interrupts generated on rising edges.
DIO_CLASS1_INT_FALLING_EDGE	The handler is to process interrupts generated on falling edges.

InterruptHandler Lets you specify the address of the function which is to handle the generated interrupt.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmIn_setTimeout

Syntax

```
UInt32 DioCl1PwmIn_setTimeout(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    Float64 TimeoutLimit)
```

Include file

`IoDrvDioClass1PwmIn.h`

Purpose

To set the timeout value for signal measurement via a DIO Class 1 PWM input channel.

Description

The PWM input channel waits for the signal edges which you specified by using `DioCl1PwmIn_SetUpdateMode`. The standard maximum waiting time is given by the lower limit of the measuring range used for PWM signal measurement. For example, if you are measuring PWM input signals that have a frequency of up to 0.5 Hz as the lower limit and a duty cycle of 50%, the waiting time until a new measuring value is available is maximal 1 second (when updating on both edges).

If you want to decrease the standard timeout, for example, to detect specific input signals faster, you can specify a lower timeout limit by using `DioCl1PwmIn_setTimeout`.

Setting a timeout limit therefore increases the lower limit of the standard measuring range for PWM input signals.

If the input signal does not change its state or if its frequency is below the lower limit of the measuring range for the standard or the specified timeout, the duty cycle results in 0% or 100% (depending on the current constant level) and the frequency is set to 0.0 Hz.

These settings do not take effect until you call `DioCl1PwmIn_apply` during the initialization phase.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using `DioCl1PwmIn_create`.

TimeoutLimit Lets you specify the time interval the PWM input channel waits for a new edge in seconds in the range 50 µs ... 1 s. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmIn_setTriggerMode

Syntax

```
UInt32 DioCl1PwmIn_setTriggerMode(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    UInt32 TriggerMode)
```

Include file

`IoDrvDioClass1PwmIn.h`

Purpose

To set the trigger mode of a DIO Class 1 PWM input channel.

Description	<p>This function is used to generate trigger signals on rising, falling or both edges of the input signal connected to the specified channel. Additionally, you have to specify the trigger line for the generated trigger signal by using DioCl1PwmIn_setTriggerLineOut. The I/O unit that is to consume the trigger signal has to specify this trigger line by using its associated xxx_setTriggerLineIn function.</p> <p>If you want to configure a combination of interrupts and trigger signals, you have to configure the interrupt mode in addition to the trigger mode by using DioCl1PwmIn_setInterruptMode.</p> <p>The setting does not take effect until you have called DioCl1PwmIn_apply during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>								
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Measurement (DIO Class 1) (MicroLabBox Features).</p>								
Parameters	<p>pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using DioCl1PwmIn_create.</p> <p>TriggerMode Lets you specify the trigger mode. The symbols can be combined by using the bitwise OR operator.</p>								
	<table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_NO_TRIGGER</td><td>No trigger is to be generated (default).</td></tr> <tr> <td>DIO_CLASS1_TRIG_RISING_EDGE</td><td>Triggers are generated on rising edges.</td></tr> <tr> <td>DIO_CLASS1_TRIG_FALLING_EDGE</td><td>Triggers are generated on falling edges.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_NO_TRIGGER	No trigger is to be generated (default).	DIO_CLASS1_TRIG_RISING_EDGE	Triggers are generated on rising edges.	DIO_CLASS1_TRIG_FALLING_EDGE	Triggers are generated on falling edges.
Symbol	Meaning								
DIO_CLASS1_NO_TRIGGER	No trigger is to be generated (default).								
DIO_CLASS1_TRIG_RISING_EDGE	Triggers are generated on rising edges.								
DIO_CLASS1_TRIG_FALLING_EDGE	Triggers are generated on falling edges.								
Return value	The function returns an error code.								
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.		
Error Code	Meaning								
IOLIB_NO_ERROR	The function was successfully completed.								
!= IOLIB_NO_ERROR	The function was not completed.								

DioCl1PwmIn_setTriggerLineOut

Syntax

```
UInt32 DioCl1PwmIn_setTriggerLineOut(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    UInt32 TriggerLineOut)
```

Include file

`IoDrvDioClass1PwmIn.h`

Purpose

To select the trigger line used for trigger signal generation.

Description

A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board. You have to specify the trigger line the consumer must listen to by using the relevant `xxx_setTriggerLineIn` function. A trigger line can be allocated only once, but multiple consumers can listen to it.

Before you can use trigger signals, you have to set the trigger mode by using `DioCl1PwmIn_setTriggerMode`. With `DioCl1PwmIn_setTriggerLineOutStatus` you enable or disable the trigger line.

The setting does not take effect until you have called `DioCl1PwmIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using `DioCl1PwmIn_create`.

TriggerLineOut Lets you specify the number of the trigger line to be used for transmitting the trigger signal in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.
...	...
DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**Basics**

[DioCl1PwmIn_setTriggerLineOutStatus](#)..... 361

DioCl1PwmIn_setTriggerLineOutStatus

Syntax

```
UInt32 DioCl1PwmIn_setTriggerLineOutStatus(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    UInt32 TriggerLineStatus)
```

Include file

`IoDrvDioClass1PwmIn.h`

Purpose

To set the status of the trigger line.

Description

With this function, you can enable or disable the trigger line that you specified before by using `DioCl1PwmIn_setTriggerLineOut`.

The function is intended to be called during the initialization phase of the real-time application or during a run-to-stop or a stop-to-run transition of the real-time application.

The setting does not take effect until you call `DioCl1PwmIn_start`.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using **DioCl1PwmIn_create**.

TriggerLineStatus Lets you specify the status of the trigger line.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_DISABLE	Disables the trigger line.
DIO_CLASS1_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1PwmIn_create.....	345
DioCl1PwmIn_start.....	363

DioCl1PwmIn_setUpdateMode

Syntax

```
UInt32 DioCl1PwmIn_setUpdateMode(
    DioCl1PwmInSDrvObject *pDioCl1PwmIn,
    UInt32 UpdateMode)
```

Include file

IoDrvDioClass1PwmIn.h

Purpose

To set the update mode of the DIO Class 1 PWM input channel.

Description

With this function you specify the edge type used for updating the value of the PWM input channel.

For further information on the update mode, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

The setting does not take effect until you have called **DioCl1PwmIn_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [PWM Signal Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using [DioCl1PwmIn_create](#).

UpdateMode Lets you select the update mode used when measuring a PWM input signal.

Symbol	Meaning
DIO_PWM_IN_RISING_EDGE	The measured values of the PWM input channel are updated at every rising edge of the PWM input signal (default).
DIO_PWM_IN_FALLING_EDGE	The measured values of the PWM input channel are updated at every falling edge of the PWM input signal.
DIO_PWM_IN_ANY_EDGE	The measured values of the PWM input channel are updated at both edges of the PWM input signal.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PwmIn_start

Syntax

```
UInt32 DioCl1PwmIn_start(DioCl1PwmInSDrvObject *pDioCl1PwmIn)
```

Include file

`IoDrvDioClass1PwmIn.h`

Purpose

To start the DIO Class 1 PWM input channel.

Description	<p>This function starts the specified DioCl1PwmIn driver so that it will activate its digital input channel. The digital input channel is enabled to start capturing the PWM input signal. This function also enables the configured trigger lines. If you call <code>DioCl1PwmIn_setTriggerLineOutStatus(DIO_CLASS1_TRIGGER_LINE_DISABLE)</code> before, the trigger line stays disabled.</p> <p>The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to PWM Signal Measurement (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PwmIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PwmIn driver object had to be created before by using <code>DioCl1PwmIn_create</code>.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics**Basics**

DioCl1PwmIn_setTriggerLineOutStatus.....	361
--	-----

Pulse Signal Generation (Pulse Out)

Introduction

MicroLabBox provides functions to generate pulse signals.

For details, refer to [Pulse Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Where to go from here	Information in this section
	DioCl1PulseOut_apply 365 To apply the initialization settings to the DIO Class 1 pulse output channel.
	DioCl1PulseOut_create 366 To create the I/O driver object for pulse signal generation using a DIO Class 1 output channel.
	DioCl1PulseOut_setFirePulse 368 To generate a pulse signal by software to be output on a DIO Class 1 pulse output channel.
	DioCl1PulseOut_setInvertingMode 369 To set the inverting mode of the DIO Class 1 pulse output channel.
	DioCl1PulseOut_setOutputHighZ 370 To set the high impedance state of the DIO Class 1 pulse output channel.
	DioCl1PulseOut_setPulseWidth 371 To set the pulse width of the DIO Class 1 pulse output channel.
	DioCl1PulseOut_setSignalVoltage 372 To set the voltage level used for the DIO Class 1 pulse output channel.
	DioCl1PulseOut_setTriggerLineIn 373 To set the trigger line that controls the DIO Class 1 pulse out channel.
	DioCl1PulseOut_start 374 To start the pulse signal generation on a DIO Class 1 pulse output channel.
	DioCl1PulseOut_write 375 To update the data of the specified DIO Class 1 pulse output channel during run time.

DioCl1PulseOut_apply

Syntax	<pre>UInt32 DioCl1PulseOut_apply(DioCl1PulseOutSDrvObject *pDioCl1PulseOut)</pre>
Include file	<code>IoDrvDioClass1PulseOut.h</code>
Purpose	To apply the initialization settings to the DIO Class 1 pulse output channel.

Description	<p>Before you can start the specified output channel of the DIO Class 1 unit, you have to transfer the settings of the DioCl1PulseOut driver object to its associated output channel by using DioCl1PulseOut_apply. You have to do this also for the default values and for values that you specified by using the DioCl1PulseOut_set<Parameter> functions. The DioCl1PulseOut_apply function is intended to be called at the end of the initialization phase of the real-time application.</p> <p>The signal generation on the specified digital output channel is not activated until you have called DioCl1PulseOut_start.</p> <p>To update settings during run time, you have to use DioCl1PulseOut_write. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Pulse Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PulseOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseOut driver object had to be created before by using DioCl1PulseOut_create.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PulseOut_create

Syntax	<pre>UInt32 DioCl1PulseOut_create(DioCl1PulseOutSDrvObject **ppDioCl1PulseOut, UInt32 Port, UInt32 Channel)</pre>
---------------	--

Include file	<code>IoDrvDioClass1PulseOut.h</code>
---------------------	---------------------------------------

Purpose	To create the I/O driver object for pulse signal generation using a DIO Class 1 output channel.
----------------	---

Description	<p>This function performs all the steps necessary to create an I/O driver object for the DIO Class 1 unit. A DioCl1PulseOut driver object handles one digital output channel for pulse signal generation. Pulse signals are used, for example, for external trigger signals controlled by a trigger line of the I/O board.</p> <p>The specified DIO Class 1 channel is initialized with default values so that it is ready for use.</p> <p>The initial values are:</p> <ul style="list-style-type: none"> ▪ Output signal voltage level: 2.5 V ▪ Pulse width: 50 µs ▪ Inverting mode: do not invert output signal ▪ Rising edge delay: 0 s <p>The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>
--------------------	---

I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Pulse Signal Generation (DIO Class 1) (MicroLabBox Features).</p>
--------------------	--

Parameters	<p>ppDioCl1PulseOut Lets you specify the address of a variable which holds the address of the created driver object.</p> <p>Port Lets you specify the number of the DIO Class 1 port providing the required channels in the range 1 ... 3.</p>
-------------------	--

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
DIO_CLASS1_PORT_2	Specifies port 2 of the DIO Class 1 unit.
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseOut_setFirePulse

Syntax

```
UInt32 DioCl1PulseOut_setFirePulse(
    DioCl1PulseOutSDrvObject *pDioCl1PulseOut)
```

Include file

`IoDrvDioClass1PulseOut.h`

Purpose

To generate a pulse signal by software to be output on a DIO Class 1 pulse output channel.

Description

If the generation of a pulse is not triggered by an event transmitted via trigger line, you can generate it via software. This function only specifies the generation of a pulse signal but the signal will not be output immediately at the specified pulse output channel.

If a trigger line is specified and applied to be used for generating a pulse, calling `DioCl1PulseOut_setFirePulse` is ignored.

The setting does not take effect until you call `DioCl1PulseOut_write` during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the pulse output that you specified by using `DioCl1PulseOut_create`, you must use `DioCl1PulseOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters	pDioCl1PulseOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseOut driver object had to be created before by using DioCl1PulseOut_create .
-------------------	---

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PulseOut_setInvertingMode

Syntax	<pre>UInt32 DioCl1PulseOut_setInvertingMode(DioCl1PulseOutSDrvObject *pDioCl1PulseOut, UInt32 InvertingMode)</pre>
Include file	<code>IoDrvDioClass1PulseOut.h</code>
Purpose	To set the inverting mode of the DIO Class 1 pulse output channel.
Description	<p>With this function, you can specify whether the pulse output signal is to be inverted at the output channel.</p> <p>The setting takes effect not before you have called DioCl1PulseOut_apply during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Pulse Signal Generation (DIO Class 1) (MicroLabBox Features).</p>
Parameters	pDioCl1PulseOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseOut driver object had to be created before by using DioCl1PulseOut_create .

InvertingMode Lets you select the inverting mode used when generating an output signal.

Symbol	Meaning
DIO_CLASS1_NOT_INVERTED	The output signal is not to be inverted (default).
DIO_CLASS1_INVERTED	The output signal is to be inverted.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseOut_setOutputHighZ

Syntax

```
UInt32 DioCl1PulseOut_setOutputHighZ(
    DioCl1PulseOutSDrvObject *pDioCl1PulseOut,
    UInt32 HighZStateOnOff)
```

Include file

IoDrvDioClass1PulseOut.h

Purpose

To set the high impedance state of the DIO Class 1 pulse output channel.

Description

You can force the output channel of the driver object to the high impedance state, for example, for termination purposes, by setting the high impedance state to *On*. If you want the real-time application to control the output levels, you can release the high impedance state again.

The setting takes effect only after you call **DioCl1PulseOut_start** or **DioCl1PwmOut_write** during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PulseOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseOut driver object had to be created before by using **DioCl1PulseOut_create**.

HighZStateOnOff Lets you specify if outputs are to be set to the high impedance state or if outputs are to be released from the high impedance state.

Symbol	Meaning
DIO_CLASS1_HIGH_Z_OFF	Releases the high impedance state.
DIO_CLASS1_HIGH_Z_ON	Sets outputs to the high impedance state (default).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseOut_setPulseWidth

Syntax

```
UInt32 DioCl1PulseOut_setPulseWidth(
    DioCl1PulseOutSDrvObject *pDioCl1PulseOut,
    Float64 PulseWidth)
```

Include file

IoDrvDioClass1PulseOut.h

Purpose

To set the pulse width of the DIO Class 1 pulse output channel.

Description

The specified value for the pulse width is stored in an internal buffer of the driver object and is not immediately updated at the output channel.

The setting does not take effect until you call **DioCl1PwmOut_apply** during the initialization phase or **DioCl1PwmOut_write** during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the pulse output that you specified by using **DioCl1PulseOut_create**, you must use **DioCl1PulseOut_start**.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) .
	For more information on the I/O mapping, refer to Pulse Signal Generation (DIO Class 1) (MicroLabBox Features) .
Parameters	<p>pDioCl1PulseOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseOut driver object had to be created before by using DioCl1PulseOut_create.</p> <p>PulseWidth Lets you specify the pulse width of the signal to be generated in seconds in the range 100 ns ... 0.5 s. You can specify values in steps of 10 ns.</p>
Return value	The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseOut_setSignalVoltage

Syntax	<pre>UInt32 DioCl1PulseOut_setSignalVoltage(DioCl1PulseOutSDrvObject *pDioCl1PulseOut, UInt32 SignalVoltageSelector)</pre>
Include file	<code>IoDrvDioClass1PulseOut.h</code>
Purpose	To set the voltage level used for the DIO Class 1 pulse output channel.
Description	<p>The output voltage level will be applied to the pulse output channel that is controlled by the specified DioCl1PulseOut driver object.</p> <p>The setting takes effect not before you have called DioCl1PulseOut_apply during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration ).								
	For more information on the I/O mapping, refer to Pulse Signal Generation (DIO Class 1) (MicroLabBox Features ).								
Parameters	<p>pDioCl1PulseOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseOut driver object had to be created before by using DioCl1PulseOut_create.</p> <p>SignalVoltageSelector Lets you select the voltage level used for the output signal(s).</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_SIGNAL_2_5_V</td><td>Specifies 2.5 V as the output voltage level (default).</td></tr> <tr> <td>DIO_CLASS1_SIGNAL_3_3_V</td><td>Specifies 3.3 V as the output voltage level.</td></tr> <tr> <td>DIO_CLASS1_SIGNAL_5_0_V</td><td>Specifies 5.0 V as the output voltage level.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_SIGNAL_2_5_V	Specifies 2.5 V as the output voltage level (default).	DIO_CLASS1_SIGNAL_3_3_V	Specifies 3.3 V as the output voltage level.	DIO_CLASS1_SIGNAL_5_0_V	Specifies 5.0 V as the output voltage level.
Symbol	Meaning								
DIO_CLASS1_SIGNAL_2_5_V	Specifies 2.5 V as the output voltage level (default).								
DIO_CLASS1_SIGNAL_3_3_V	Specifies 3.3 V as the output voltage level.								
DIO_CLASS1_SIGNAL_5_0_V	Specifies 5.0 V as the output voltage level.								
Return value	The function returns an error code.								
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.		
Error Code	Meaning								
IOLIB_NO_ERROR	The function was successfully completed.								
!= IOLIB_NO_ERROR	The function was not completed.								

DioCl1PulseOut_setTriggerLineIn

Syntax	<pre>UInt32 DioCl1PulseOut_setTriggerLineIn(DioCl1PulseOutSDrvObject *pDioCl1PulseOut, UInt32 TriggerLineIn)</pre>
Include file	<code>IoDrvDioClass1PulseOut.h</code>
Purpose	To set the trigger line that controls the DIO Class 1 pulse out channel.
Description	The generation of a pulse signal can be controlled by software using the DioCl1PulseOut_setFirePulse function or by an internal trigger signal. If you want to use an internal trigger signal, you have to specify the trigger line which the pulse out channel has to listen to.

The setting takes effect not before you have called **DioCl1PulseOut_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PulseOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseOut driver object had to be created before by using [DioCl1PulseOut_create](#).

TriggerLineIn Lets you specify the number of the trigger line that controls generating the pulse signal in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1 of the DIO Class 1 unit.
...	...
DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16 of the DIO Class 1 unit.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseOut_start

Syntax

```
UInt32 DioCl1PulseOut_start(
    DioCl1PulseOutSDrvObject *pDioCl1PulseOut)
```

Include file

`IoDrvDioClass1PulseOut.h`

Purpose

To start the pulse signal generation on a DIO Class 1 pulse output channel.

Description

This function starts the specified DioCl1PulseOut driver so that it will activate its digital output channel. The digital output channel is enabled to generate the output signal. If you call **DioCl1PulseOut_setOutputHighZ(DIO_CLASS1_HIGH_Z_ON)** before, the output channel stays in the high impedance state.

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the pulse output that you specified by using **DioCl1PulseOut_create**, you must use **DioCl1PulseOut_start**.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PulseOut Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseOut driver object had to be created before by using **DioCl1PulseOut_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

[DioCl1PulseOut_setOutputHighZ.....](#) 370

DioCl1PulseOut_write

Syntax

```
UInt32 DioCl1PulseOut_write(
    DioCl1PulseOutSDrvObject *pDioCl1PulseOut)
```

Include file	<code>IoDrvDioClass1PulseOut.h</code>						
Purpose	To update the data of the specified DIO Class 1 pulse output channel during run time.						
Description	<p>If have used one of the following functions during run time, their settings will not be updated on the hardware until you have called <code>DioCl1PulseOut_write</code>.</p> <ul style="list-style-type: none"> ▪ <code>DioCl1PulseOut_setPulseWidth</code> ▪ <code>DioCl1PulseOut_setOutputHighZ</code> ▪ <code>DioCl1PulseOut_setFirePulse</code> <p>The <code>DioCl1PulseOut_write</code> function is also required to output the specified pulse signal at the pulse output channel if you trigger the pulse signal via software by using <code>DioCl1PulseOut_setFirePulse</code>.</p> <p>The specified digital output channel must be enabled by using <code>DioCl1PulseOut_start</code>.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Pulse Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p><code>pDioCl1PulseOut</code> Lets you specify the DIO Class 1 channel to be used. The <code>DioCl1PulseOut</code> driver object had to be created before by using <code>DioCl1PulseOut_create</code>.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td> <td>The function was successfully completed.</td> </tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						

Pulse Width Measurement (Pulse In)

Introduction MicroLabBox provides functions to measure the width of a pulse signal.

For details, refer to [Pulse Width Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Where to go from here

Information in this section

DioCl1PulseIn_apply	378
To apply the initialization settings to the DIO Class 1 pulse input channel.	
DioCl1PulseIn_create	379
To create the I/O driver object for pulse width measurement using a DIO Class 1 input channel.	
DioCl1PulseIn_disableInterrupts	380
To disable generation of interrupts on a DIO Class 1 pulse input channel.	
DioCl1PulseIn_enableInterrupts	381
To enable generation of interrupts on a DIO Class 1 pulse input channel.	
DioCl1PulseIn_getPulseWidth	382
To get the pulse width of the square-wave signal captured by a DIO Class 1 pulse input channel.	
DioCl1PulseIn_read	384
To read the data from a DIO Class 1 pulse input channel.	
DioCl1PulseIn_setEdgePolarity	385
To set the edge polarity of the DIO Class 1 pulse input channel.	
DioCl1PulseIn_setEventDelay	386
To set the delay value for events generated by a DIO Class 1 pulse input channel.	
DioCl1PulseIn_setInterruptMode	387
To set the interrupt mode of a DIO Class 1 pulse input channel.	
DioCl1PulseIn_setIntVector	388
To set the interrupt handler for processing interrupts from a DIO Class 1 pulse input channel.	
DioCl1PulseIn_setTriggerMode	390
To set the trigger mode of a DIO Class 1 pulse input channel.	
DioCl1PulseIn_setTriggerLineOut	391
To set the trigger line used for trigger signals on specific condition on DIO Class 1 pulse input channel.	
DioCl1PulseIn_setTriggerLineOutStatus	392
To set the status of the trigger line.	
DioCl1PulseIn_setWidthMax	393
To set the maximum pulse width for a DIO Class 1 pulse input channel.	
DioCl1PulseIn_start	394
To start the DIO Class 1 pulse input channel.	

DioCl1PulseIn_apply

Syntax	<pre>UInt32 DioCl1PulseIn_apply(DioCl1PulseInSDrvObject *pDioCl1PulseIn)</pre>						
Include file	<code>IoDrvDioClass1PulseIn.h</code>						
Purpose	To apply the initialization settings to the DIO Class 1 pulse input channel.						
Description	<p>Before you can start the specified input channel of the DIO Class 1 unit, you have to transfer the settings of the DioCl1PulseIn driver object to its related input channel by using <code>DioCl1PulseIn_apply</code>. You have to do this also for the default values and for values that you specified by using the <code>DioCl1PulseIn_set<Parameter></code> functions. The <code>DioCl1PulseIn_apply</code> function is intended to be called at the end of the initialization phase of the real-time application.</p> <p>The signal measurement on the specified digital input channel is activated not before you have called <code>DioCl1PulseIn_start</code>.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Pulse Width Measurement (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using <code>DioCl1PulseIn_create</code>.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PulseIn_create

Syntax

```
UInt32 DioCl1PulseIn_create(
    DioCl1PulseInSDrvObject **ppDioCl1PulseIn,
    UInt32 Port,
    UInt32 Channel)
```

Include file

`IoDrvDioClass1PulseIn.h`

Purpose

To create the I/O driver object for pulse width measurement using a DIO Class 1 input channel.

Description

This function performs all the steps necessary to create an I/O driver object for the DIO Class 1 unit. A `DioCl1PulseIn` driver object handles one digital input channel for measuring the pulse width of a pulse signal.

The specified DIO Class 1 channel is initialized with default values so that it is ready for use.

The initial values are:

- Edge polarity: rising edge
- Pulse width maximum: 1.34 s
- Interrupt mode: no interrupt generation
- Trigger mode: no trigger signal generation
- Event downsample: 1
- Event delay: 0 s

The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Width Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

ppDioCl1PulseIn Lets you specify the address of a variable which holds the address of the created driver object.

Port Lets you specify the number of the DIO Class 1 port providing the required channels in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
DIO_CLASS1_PORT_2	Specifies port 2 of the DIO Class 1 unit.
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

Channel Lets you specify the channel to be used in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified port of the DIO Class 1 unit.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseIn_disableInterrupts

Syntax

```
UInt32 DioCl1PulseIn_disableInterrupts(
    DioCl1PulseInSDrvObject *pDioCl1PulseIn,
    UInt32 InterruptSelect)
```

Include file

IoDrvDioClass1PulseIn.h

Purpose

To disable generation of interrupts on a DIO Class 1 pulse input channel.

Description

With this function, you can disable the generation of interrupts that you specified before by using **DioCl1PulseIn_setInterruptMode**. The settings take effect immediately.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration ).				
	For more information on the I/O mapping, refer to Pulse Width Measurement (DIO Class 1) (MicroLabBox Features ).				
Parameters	<p>pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using DioCl1PulseIn_create.</p> <p>InterruptSelect Lets you select the interrupts that are to be disabled.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_INT_DATA_READY</td><td>Disables the generation of interrupts if a new value is available.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_INT_DATA_READY	Disables the generation of interrupts if a new value is available.
Symbol	Meaning				
DIO_CLASS1_INT_DATA_READY	Disables the generation of interrupts if a new value is available.				
Return value	The function returns an error code.				
Error Code	Meaning				
IOLIB_NO_ERROR	The function was successfully completed.				
!= IOLIB_NO_ERROR	The function was not completed.				

DioCl1PulseIn_enableInterrupts

Syntax	<pre>UInt32 DioCl1PulseIn_enableInterrupts(DioCl1PulseInSDrvObject *pDioCl1PulseIn, UInt32 InterruptSelect)</pre>
Include file	<code>IoDrvDioClass1PulseIn.h</code>
Purpose	To enable the generation of interrupts on a DIO Class 1 pulse input channel.
Description	<p>With this function, you can either enable the generation of interrupts or reenable the generation of interrupts that you disabled before by using DioCl1PulseIn_disableInterrupts. The interrupts have to be specified before by using DioCl1PulseIn_setInterruptMode. The settings take effect immediately.</p> <p>After calling DioCl1PulseIn_start, interrupts are disabled.</p>

The function is intended to be called during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Width Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using [DioCl1PulseIn_create](#).

InterruptSelect Lets you select the interrupts that are to be enabled.

Symbol	Meaning
DIO_CLASS1_INT_DATA_READY	Enables the generation of interrupts if a new value is available.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1PulseIn_disableInterrupts.....	380
DioCl1PulseIn_setInterruptMode.....	387

DioCl1PulseIn_getPulseWidth

Syntax

```
UInt32 DioCl1PulseIn_getPulseWidth(
    DioCl1PulseInSDrvObject *pDioCl1PulseIn,
    Float64 *pPulseWidth)
```

Include file

IoDrvDioClass1PulseIn.h

Purpose	To get the pulse width of the square-wave signal captured by a DIO Class 1 pulse input channel.
----------------	---

Description	<p>Because you get the values from the internal buffer of the driver object, there might be newer input values. To update the internal buffer with the latest values, you first have to call DioCl1PulseIn_read.</p> <p>The pulse width measurement supports measuring input signals with a duration of 50 ns up to 1.34 s of the specified edge polarity, refer to DioCl1PulseIn_setEdgePolarity.</p>
--------------------	--

Note

- The pulse width can be correctly measured only if both the high time and the low time are at least 50 ns. If the high time or the low time are below this minimum value, the resulting value of the pulse width cannot be predicted.
- If the pulse width of the low time or the high time of the input signal exceeds the maximum of 1.34 s, the function outputs the maximum value of the Float64 data type (DBL_MAX).

Note

All digital inputs of the DIO Class 1 unit are in a disabled state after reset and will read 0. To enable the pulse input channel that you specified by using [DioCl1PulseIn_create](#), you must use [DioCl1PulseIn_start](#).

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration ).
--------------------	--

For more information on the I/O mapping, refer to [Pulse Width Measurement \(DIO Class 1\)](#) ([MicroLabBox Features](#) ).

Parameters	<p>pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using DioCl1PulseIn_create.</p> <p>pPulseWidth Lets you specify the address of a variable that holds the measured pulse width in seconds in the range 50 ns ... 1.34 s with a resolution of 10 ns.</p>
-------------------	--

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PulseIn_read

Syntax	<pre>UInt32 DioCl1PulseIn_read(DioCl1PulseInSDrvObject *pDioCl1PulseIn)</pre>
Include file	<code>IoDrvDioClass1PulseIn.h</code>
Purpose	To read the data from a DIO Class 1 pulse input channel.
Description	<p>This function reads the raw data of the pulse input signal captured by the specified pulse input channel. It calculates the pulse width of the specified edge polarity and stores the result in an internal buffer of the driver object.</p> <p>The input channel must have been enabled before by using DioCl1PulseIn_start.</p> <p>To get the measured value from the internal buffer, you must use DioCl1PulseIn_getPulseWidth.</p> <p>The function is intended to be used during run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Pulse Width Measurement (DIO Class 1) (MicroLabBox Features).</p>
Parameters	<p>pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using DioCl1PulseIn_create.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseIn_setEdgePolarity

Syntax

```
UInt32 DioCl1PulseIn_setEdgePolarity(
    DioCl1PulseInSDrvObject *pDioCl1PulseIn,
    UInt32 EdgePolarity)
```

Include file

`IoDrvDioClass1PulseIn.h`

Purpose

To set the edge polarity of the DIO Class 1 pulse input channel.

Description

This function is used to specify whether the high-level section or the low-level section of the square-wave signal is to be used for measuring the pulse width. The section to be used is defined by the specified edge polarity.

- Rising edge

The duration between a rising edge and the next falling edge of the square-wave input signal is measured.

- Falling edge

The duration between a falling edge and the next rising edge of the square-wave input signal is measured.

The setting does not take effect until you have called `DioCl1PulseIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Width Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using **DioCl1PulseIn_create**.

EdgePolarity Lets you specify the edge for starting the measurement.

Symbol	Meaning
DIO_PW2D_RISING_EDGE	Measurement starts at rising edges (default).
DIO_PW2D_FALLING_EDGE	Measurement starts at falling edges.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseIn_setEventDelay

Syntax

```
UInt32 DioCl1PulseIn_setEventDelay(
    DioCl1PulseInSDrvObject *pDioCl1PulseIn,
    Float64 EventDelay)
```

Include file

IoDrvDioClass1PulseIn.h

Purpose To set the delay value for events generated by a DIO Class 1 pulse input channel.

Description

The function is used to delay the generation of events that you specified by using **DioCl1PulseIn_setInterruptMode** and **DioCl1PulseIn_setTriggerMode**. With these functions, you also specified the channel that is affected by the delay setting.

The delay is the time interval between the occurrence of the event and the generation of the event signal (interrupt or trigger signal).

The setting does not take effect until you have called **DioCl1PulseIn_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Pulse Width Measurement](#) ([DIO Class 1](#)) ([MicroLabBox Features](#)).

Parameters

pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using [DioCl1PulseIn_create](#).

EventDelay Lets you specify the time interval by which the generation of an event is delayed in seconds in the range 0 ... 1.34 s. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseIn_setInterruptMode

Syntax

```
UInt32 DioCl1PulseIn_setInterruptMode(
    DioCl1PulseInSDrvObject *pDioCl1PulseIn,
    UInt32 InterruptMode)
```

Include file

`IoDrvDioClass1PulseIn.h`

Purpose

To set the interrupt mode of a DIO Class 1 pulse input channel.

Description

This function is used to generate a *data ready* interrupt if a new pulse width value has been measured on the specified input channel.

If you want to configure a combination of interrupts and trigger signals, you have to configure the trigger mode in addition to the interrupt mode by using [DioCl1PulseIn_setTriggerMode](#).

The setting does not take effect until you have called [DioCl1PulseIn_apply](#) during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Width Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using [DioCl1PulseIn_create](#).

InterruptMode Lets you specify the interrupt mode.

Symbol	Meaning
DIO_CLASS1_NO_INTERRUPT	No interrupt is to be generated (default).
DIO_CLASS1_INT_DATA_READY	An interrupt is to be generated if a new value is available.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseIn_setIoIntVector

Syntax

```
UInt32 DioCl1PulseIn_SetIoIntVector(
    DioCl1PulseInSDrvObject *pDioCl1PulseIn,
    UInt32 InterruptSelect,
    IoLibTToIntHandler InterruptHandler)
```

Include file

`IoDrvDioClass1PulseIn.h`

Purpose

To set the interrupt handler for processing interrupts from a DIO Class 1 pulse input channel.

Description

This function is used to register the function that handles the generated interrupts specified by using `DioCl1PulseIn_setInterruptMode`.

Note

The condition for interrupt generation must be the same as specified for the `DioCl1PulseIn_setInterruptMode` function.

The setting does not take effect until you have called `DioCl1PulseIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Width Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using `DioCl1PulseIn_create`.

InterruptSelect Lets you specify the interrupt the given interrupt handler is to process.

Symbol	Meaning
DIO_CLASS1_INT_DATA_READY	The handler is to process interrupts generated if a new value is available.

InterruptHandler Lets you specify the address of the function which is to handle the generated interrupt.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseIn_setTriggerMode

Syntax

```
UInt32 DioCl1PulseIn_setTriggerMode(
    DioCl1PulseInSDrvObject *pDioCl1PulseIn,
    UInt32 TriggerMode)
```

Include file

`IoDrvDioClass1PulseIn.h`

Purpose

To set the trigger mode of a DIO Class 1 pulse input channel.

Description

This function is used to generate a trigger signal if a new pulse width value has been measured on the specified input channel. Additionally, you have to specify the trigger line for the generated trigger signal by using `DioCl1PulseIn_setTriggerLineOut`. The I/O unit that is to consume the trigger signal has to specify this trigger line by using its associated `xxx_setTriggerLineIn` function.

If you want to configure a combination of interrupts and trigger signals, you have to configure the interrupt mode in addition to the trigger mode by using `DioCl1PulseIn_setInterruptMode`.

The setting does not take effect until you have called `DioCl1PulseIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Width Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using `DioCl1PulseIn_create`.

TriggerMode Lets you specify the trigger mode.

Symbol	Meaning
<code>DIO_CLASS1_NO_TRIGGER</code>	No trigger is to be generated (default).
<code>DIO_CLASS1_TRIG_DATA_READY</code>	A trigger is to be generated if a new value is available.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1PulseIn_setTriggerLineOut

Syntax

```
UInt32 DioCl1PulseIn_setTriggerLineOut(
    DioCl1PulseInSDrvObject *pDioCl1PulseIn,
    UInt32 TriggerLineOut)
```

Include file

`IoDrvDioClass1PulseIn.h`

Purpose

To set the trigger line used for trigger signals on specific condition on DIO Class 1 pulse input channel.

Description

A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board. You have to specify the trigger line the consumer must listen to by using the relevant `xxx_setTriggerLineIn` function. A trigger line can be allocated only once, but multiple consumers can listen to it.

Before you can use trigger signals, you have to set the trigger mode by using `DioCl1PulseIn_setTriggerMode`. With `DioCl1PulseIn_setTriggerLineOutStatus` you enable or disable the trigger line.

The setting does not take effect until you have called `DioCl1PulseIn_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Width Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters	<p>pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using DioCl1PulseIn_create.</p> <p>TriggerLineOut Lets you specify the number of the trigger line to be used for transmitting the trigger signal in the range 1 ... 16.</p>								
	<table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_TRIGGER_LINE_1</td><td>Specifies trigger line 1.</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td>DIO_CLASS1_TRIGGER_LINE_16</td><td>Specifies trigger line 16.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.	DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.
Symbol	Meaning								
DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.								
...	...								
DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.								
Return value	The function returns an error code.								
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.		
Error Code	Meaning								
IOLIB_NO_ERROR	The function was successfully completed.								
!= IOLIB_NO_ERROR	The function was not completed.								

Related topics	References
	<p>DioCl1PulseIn_setTriggerLineOutStatus..... 392</p>

DioCl1PulseIn_setTriggerLineOutStatus

Syntax	<pre>UInt32 DioCl1PulseIn_setTriggerLineOutStatus(DioCl1PulseInSDrvObject *pDioCl1PulseIn, UInt32 TriggerLineStatus)</pre>
Include file	<code>IoDrvDioClass1PulseIn.h</code>
Purpose	To set the status of the trigger line.
Description	<p>With this function, you can enable or disable the trigger line that you specified before by using DioCl1PulseIn_setTriggerLineOut.</p> <p>The function is intended to be called during the initialization phase of the real-time application or during a run-to-stop or a stop-to-run transition of the real-time application.</p>

The setting does not take effect until you call **DioCl1PulseIn_start**.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Pulse Width Measurement](#) ([DIO Class 1](#)) ([MicroLabBox Features](#)).

Parameters

pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using [DioCl1PulseIn_create](#).

TriggerLineStatus Lets you specify the status of the trigger line.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_DISABLE	Disables the trigger line.
DIO_CLASS1_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1PulseIn_create.....	379
DioCl1PulseIn_start.....	394

DioCl1PulseIn_setWidthMax

Syntax

```
UInt32 DioCl1PulseIn_setWidthMax(
    DioCl1PulseInSDrvObject *pDioCl1PulseIn,
    Float64 PulseWidthLimit)
```

Include file

[IoDrvDioClass1PulseIn.h](#)

Purpose	To set the maximum pulse width for a DIO Class 1 pulse input channel.						
Description	<p>The pulse width measurement is based on edge detection. The standard maximum waiting time interval is the upper limit of the pulse width range of 1.34 s. You can use <code>DioCl1PulseIn_setWidthMax</code> to decrease this time interval by specifying an individual maximum pulse width. Then, an overflow will be detected faster. An overflow will be indicated if the input signal does not change its state or exceeds the upper limit of the pulse width range.</p> <p>The setting does not take effect until you have called <code>DioCl1PulseIn_apply</code> during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Pulse Width Measurement (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The <code>DioCl1PulseIn</code> driver object must have been created before by using <code>DioCl1PulseIn_create</code>.</p> <p>PulseWidthLimit Lets you specify the maximum pulse width to be measured in seconds in the range 50 µs ... 1.34 s. You can specify the value in steps of 10 ns.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1PulseIn_start

Syntax	<code>UInt32 DioCl1PulseIn_start(DioCl1PulseInSDrvObject *pDioCl1PulseIn)</code>
Include file	<code>IoDrvDioClass1PulseIn.h</code>

Purpose

To start the DIO Class 1 pulse input channel.

Description

This function starts the specified DioCl1PulseIn driver so that it will activate its digital input channel. The digital input channel is enabled to start capturing the pulse input signal. This function also enables the configured trigger line. If you call

`DioCl1PulseIn_setTriggerLineOutStatus(DIO_CLASS1_TRIGGER_LINE_DISABLE)` before, the trigger line stays disabled.

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Pulse Width Measurement \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1PulseIn Lets you specify the DIO Class 1 channel to be used. The DioCl1PulseIn driver object must have been created before by using `DioCl1PulseIn_create`.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

`DioCl1PulseIn_setTriggerLineOutStatus.....` 392

USB Flight Recorder

Purpose

With the USB Flight Recorder, you can perform long-term data acquisition. During the simulation, the values of selectable variables are written to the connected USB mass storage device. The available storage size is only restricted by the USB mass storage device.

Introduction to the USB Flight Recorder

Purpose

With the USB Flight Recorder, you can perform long-term data acquisition. During the simulation, the values of selectable variables are written to the connected USB mass storage device.

Description

Any standard USB mass storage device can be used, such as a USB memory stick or an external USB hard drive with or without separate power supply. The USB device must be formatted with the Microsoft FAT32 file system and must be directly connected to MicroLabBox. Connection via USB hubs is not supported.

The recorded data is written to a sequence of files stored in the root directory of the USB device. The file names are generated automatically and contain the name of the real-time model as well as the creation date and time of the file. Only one file is written at a time. The file grows until it reaches a user-defined maximum size (refer to [dsflrec_usb_initialize \(USB Flight Recorder RTLib Reference\)](#)). After that, the file is closed and a new output file is created. As long as the USB Flight Recording session runs, new files are generated until the maximum number of files has been reached. The maximum file number is given by the size of the USB device divided by the maximum file size.

When the maximum number of files is reached, either the oldest file is deleted or the USB Flight Recording session is stopped (refer to [dsflrec_usb_initialize \(USB Flight Recorder RTLib Reference\)](#)).

On multicore platforms such as MicroLabBox, the USB Flight Recorder is separately configured for each real-time application running on the board. Each instantiated USB Flight Recorder generates its own output files.

The real-time model continues to run, even if the USB Flight Recording session is stopped.

Note

To avoid data loss, use the `dsflrec_usb_eject` function before unplugging the USB memory stick or hard drive.

Characteristics

For information on the USB Flight Recorder's characteristics, such as the maximum data rate or the maximum number of variables, refer to [USB Flight Recorder \(MicroLabBox Features\)](#). The section also contains instructions on how to use the USB Flight Recorder.

Nonvolatile Data Handling (NVDATA)

Purpose	With the nonvolatile data handling (NVDATA), you can write data to the board's nonvolatile memory and read data from the memory.
----------------	--

Where to go from here	Information in this section
	NvData_apply400 To apply the initialization settings to the NvData driver object.
	NvData_create401 To create the driver object for nonvolatile data handling.
	NvData_createDataSet402 To create a data set for nonvolatile data handling.
	NvData_read403 To read a data set from the board's nonvolatile memory.
	NvData_setDimension404 To specify the number of elements in the data set.
	NvData_setName405 To specify the name of the data set.
	NvData_setType407 To specify the data type of the elements in a data set.
	NvData_write408 To write a data set to the board's nonvolatile memory.
	Example of Implementing Access to the Nonvolatile Data409 Shows you how to implement access to the board's nonvolatile data.

Information in other sections

[Nonvolatile Data Handling \(NVDATA\) \(MicroLabBox Features !\[\]\(9175fbcc28a67da77f84907a68da3a4b_img.jpg\)](#)

MicroLabBox provides access to the board's nonvolatile memory by implementing the access in a real-time application or by using the board's web interface.

[Using the Web Interface for Nonvolatile Data Handling \(MicroLabBox Features !\[\]\(1f20ab1aa6e33534417735300f1e5bfe_img.jpg\)](#)

The web interface of your MicroLabBox provides a configuration page which lets you manage the data sets stored in the board's nonvolatile memory.

NvData_apply

Syntax

```
Int32 NvData_apply(
    NvDataTDrv *pNvDataDrv)
```

Include file

NvDataRP.h

Purpose

To apply the initialization settings to the NvData driver object.

Description

Before you can use the nonvolatile data handling, you must apply the settings to the NvData driver object. The `NvData_apply` function is intended to be called at the end of the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

`pNvDataDrv` Lets you specify the address of the driver object created before by using `NvData_create`.

Return value

The function returns an error code.

Error Code	Meaning
0	The function was successfully completed.
NVDATA_ERR_DRIVER_NULL	A NULL driver object was passed to the function.
NVDATA_ERR_DUPLICATE_APPLY	The function was called more than once.

Error Code	Meaning
NVDATA_ERR_DUPLICATE_ENTRIES	Data sets with the same name but different settings were detected.
NVDATA_ERR_INTERNAL	An internal error was detected. Check the message log.

Related topics**References**

NvData_create	401
-------------------------------------	-----

NvData_create

Syntax

```
Int32 NvData_create(
    NvDataTDrv **ppNvDataDrv)
```

Include file

NvDataRP.h

Purpose

To create the driver object for nonvolatile data handling.

Description

This function performs all the steps necessary to create a driver object for nonvolatile data handling. You can create only one driver object per application. The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

ppNvDataDrv Lets you specify the address of a variable which holds the address of the created driver object.

Return value

The function returns an error code.

Error Code	Meaning
0	The function was successfully completed.
NVDATA_ERR_MAX_INSTANCE	The function was called more than once.

NvData_createDataSet

Syntax

```
Int32 NvData_createDataSet(
    NvDataTDrv *pNvDataDrv,
    UInt32 *phDataSet)
```

Include file

NvDataRP.h

Purpose

To create a data set for nonvolatile data handling.

Description

This function creates a handle for a data set. You have to completely configure the instantiated data set by specifying its name, the number of elements to be transferred, and the type of the elements.

You can create up to 64 data sets in your real-time application. In a multicore application, the sum of the data sets allocated by the subapplications also must not exceed 64.

The available memory for data sets is limited to a total of 64 KB.

Note

The board's nonvolatile memory is not automatically cleared upon application start. Data sets from previous sessions might still exist in the memory and decrease the available memory size.

- Use the board's web interface to check and manage the content of the nonvolatile memory.

The function is intended to be called during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Parameters

pNvDataDrv Lets you specify the address of the driver object created before by using **NvData_create**.

phDataSet Lets you specify the address to the created data set handler.

Return value

The function returns an error code.

Error Code	Meaning
0	The function was successfully completed.
NVDATA_ERR_OUT_OF_MEMORY	The data set cannot be created because either the number of data sets is exceeded

Error Code	Meaning
NVDATA_ERR_DRIVER_NULL	or there is not enough free nonvolatile memory available.
	The function was called with a NULL driver object.

Related topics**References**

NvData_create.....	401
--------------------	-----

NvData_read

Syntax

```
Int32 NvData_read(
    NvDataTDrv *pNvDataDrv,
    UInt32 hDataSet,
    void *pDestBuffer,
    int *DataValid)
```

Include file

NvDataRP.h

Purpose

To read a data set from the board's nonvolatile memory.

Description

You can access a data set via its handler that you created by using the **NvData_createDataSet** function. To read the data from the specified data set, you have to provide a destination buffer with at least the memory size of the specified data set. The required memory size depends on the number of elements in the data set specified by using the **NvData_setDimension** function and the data types of the contained elements specified by using the **NvData_setType** function.

The **DataValid** parameter can be used to check, if the data set has previously been written. If the data set has been successfully written at least once, **DataValid** will return 1. Otherwise it will return 0.

The function is intended to be called during the run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

Parameters

pNvDataDrv Lets you specify the address of the driver object created before by using **NvData_create**.

hDataSet Lets you specify the data set to be accessed. The data set must be created before by using `NvData_createDataSet`.

pDestBuffer Lets you specify the address of the buffer in which the read data is being stored.

The size of the buffer must match the size of the data set to be read as the result of the number of elements in the data set and their data types.

DataValid Returns a flag that shows if the specified data set is valid or not.

There must be one initial write access to the data set before a read access will be valid.

- 0: Data set has not been previously written, i.e., it is still uninitialized.
- 1: Data set has been written at least once.

Return value

The function returns an error code.

Error Code	Meaning
0	The function was successfully completed.
NVDATA_ERR_DRIVER_NULL	A NULL driver object was passed to the function.
NVDATA_ERR_INVALID_HANDLE	The <code>hDataSet</code> handle is invalid.

Related topics**References**

NvData_create.....	401
NvData_createDataSet.....	402
NvData_setDimension.....	404
NvData_setType.....	407

NvData_setDimension

Syntax

```
Int32 NvData_setDimension(
    NvDataTDrv *pNvDataDrv,
    UInt32 hDataSet,
    UInt32 Dimension)
```

Include file

NvDataRP.h

Purpose

To specify the number of elements in the data set.

Description	The memory size of a data set results from its specified dimension (number of elements) and the specified data type of the elements (see NvData_setType). The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.
Parameters	<p>pNvDataDrv Lets you specify the address of the driver object created before by using NvData_create.</p> <p>hDataSet Lets you specify the data set to be accessed. The data set must be created before by using NvData_createDataSet.</p> <p>Dimension Lets you specify the number of elements in the current data set in the range 1 ... 64.</p>
Return value	The function returns an error code.

Error Code	Meaning
0	The function was successfully completed.
NVDATA_ERR_DRIVER_NULL	A NULL driver object was passed to the function.
NVDATA_ERR_INVALID_SIZE	The specified dimension is 0.
NVDATA_ERR_TOO_MANY_ELEMENTS	The specified dimension is greater than 64.

Related topics	References
	<p>NvData_create..... 401 NvData_createDataSet..... 402 NvData_setType..... 407</p>

NvData_setName

Syntax	<pre>Int32 NvData_setName(NvDataTDrv *pNvDataDrv, UInt32 hDataSet, const char *Name)</pre>
---------------	---

Include file	NvDataRP.h								
Purpose	To specify the name of the data set.								
Description	<p>Each data set in the nonvolatile memory is uniquely identified by its name. For each data set handle that you created by using NvData_createDataSet, its name, type and dimension must be specified.</p> <p>For further information on handling the NVDATA via the board's web interface, refer to Nonvolatile Data Handling (NVDATA) (MicrolabBox Features).</p> <p>The function is intended to be called during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>								
Parameters	<p>pNvDataDrv Lets you specify the address of the driver object created before by using NvData_create.</p> <p>hDataSet Lets you specify the data set to be accessed. The data set must be created before by using NvData_createDataSet.</p> <p>Name Lets you specify a unique name for the data set with a maximum length of 63 characters.</p>								
	<p>Note</p> <p>A valid data set name is a character string of letters, digits, and underscores. There are the following naming restrictions for the data set name:</p> <ul style="list-style-type: none"> ▪ The first character must be a letter. ▪ The name must not be a keyword, such as while or if. 								
Return value	The function returns an error code.								
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>0</td><td>The function was successfully completed.</td></tr> <tr> <td>NVDATA_ERR_DRIVER_NULL</td><td>A NULL driver object was passed to the function.</td></tr> <tr> <td>NVDATA_ERR_INVALID_NAME</td><td> The specified name is invalid: <ul style="list-style-type: none"> ▪ The name is not yet specified or has a length of 0. ▪ The maximum size of 63 characters is exceeded. ▪ The specified name already exists. </td></tr> </tbody> </table>	Error Code	Meaning	0	The function was successfully completed.	NVDATA_ERR_DRIVER_NULL	A NULL driver object was passed to the function.	NVDATA_ERR_INVALID_NAME	The specified name is invalid: <ul style="list-style-type: none"> ▪ The name is not yet specified or has a length of 0. ▪ The maximum size of 63 characters is exceeded. ▪ The specified name already exists.
Error Code	Meaning								
0	The function was successfully completed.								
NVDATA_ERR_DRIVER_NULL	A NULL driver object was passed to the function.								
NVDATA_ERR_INVALID_NAME	The specified name is invalid: <ul style="list-style-type: none"> ▪ The name is not yet specified or has a length of 0. ▪ The maximum size of 63 characters is exceeded. ▪ The specified name already exists. 								

Related topics

References

NvData_create.....	401
NvData_createDataSet.....	402

NvData_SetType

Syntax

```
Int32 NvData_SetType(
    NvDataTDrv *pNvDataDrv,
    UInt32 hDataSet,
    NvDataEType DataType)
```

Include file

NvDataRP.h

Purpose

To specify the data type of the elements in a data set.

Description

The memory size of a data set results from its specified dimension (number of elements) set by **NvData_SetDimension** and the specified data type of the elements.

The function is intended to be called during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Parameters

pNvDataDrv Lets you specify the address of the driver object created before by using **NvData_create**.

hDataSet Lets you specify the data set to be accessed. The data set must be created before by using **NvData_createDataSet**.

DataType Lets you specify the data type of the elements contained in the data set. All elements in a data set must have the same data type.

The following data types are available in the **NvDataEType** enumeration.

Data Type	Meaning
NVDATA_TYPE_DOUBLE_FLOAT	Specifies 64-bit float values (8 bytes for one element)
NVDATA_TYPE_SINGLE_FLOAT	Specifies 32-bit float values (4 bytes for one element)

Data Type	Meaning
NVDATA_TYPE_UINT32	Specifies 32-bit unsigned integer values (4 bytes for one element)
NVDATA_TYPE_INT32	Specifies 32-bit signed integer values (4 bytes for one element)
NVDATA_TYPE_UINT16	Specifies 16-bit unsigned integer values (2 bytes for one element)
NVDATA_TYPE_INT16	Specifies 16-bit signed integer values (2 bytes for one element)
NVDATA_TYPE_UINT8	Specifies 8-bit unsigned integer values (1 byte for one element)
NVDATA_TYPE_INT8	Specifies 8-bit signed integer values (1 byte for one element)

Return value

The function returns an error code.

Error Code	Meaning
0	The function was successfully completed.
NVDATA_ERR_DRIVER_NULL	A NULL driver object was passed to the function.
ERR_NVTABLE_INVALID_PARAM	The specified data type is invalid. Refer to the description of the DataType parameter for the valid values.

Related topics**References**

NvData_create.....	401
NvData_createDataSet.....	402
NvData_setDimension.....	404

NvData_write

Syntax

```
Int32 NvData_write(
    NvDataTDrv *pNvDataDrv,
    UInt32 hDataSet,
    const void *pSrcBuffer)
```

Include file

NvDataRP.h

Purpose	To write a data set to the board's nonvolatile memory.
Description	<p>You can access a data set via its handler that you created by using the NvData_createDataSet function. The specified data set has to provide at least the memory size of the specified source buffer.</p> <p>The memory size of the data set depends on the specified number of elements and the data types of the contained elements, refer to NvData_setDimension and NvData_setType.</p> <p>The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.</p>
Parameters	<p>pNvDataDrv Lets you specify the address of the driver object created before by using NvData_create.</p> <p>hDataSet Lets you specify the data set to be accessed. The data set must be created before by using NvData_createDataSet.</p> <p>pSrcBuffer Lets you specify the address of the buffer providing the data that you want to write to the board's nonvolatile memory.</p>
Return value	The function returns an error code.

Related topics	References								
	<table> <tr> <td>NvData_create.....</td> <td>401</td> </tr> <tr> <td>NvData_createDataSet.....</td> <td>402</td> </tr> <tr> <td>NvData_setDimension.....</td> <td>404</td> </tr> <tr> <td>NvData_setType.....</td> <td>407</td> </tr> </table>	NvData_create	401	NvData_createDataSet	402	NvData_setDimension	404	NvData_setType	407
NvData_create	401								
NvData_createDataSet	402								
NvData_setDimension	404								
NvData_setType	407								

Example of Implementing Access to the Nonvolatile Data

Introduction	The following example code shows you the order in which the NvData functions are to be used.
---------------------	--

Using NvData functions

The example code shows you how to create and configure two data sets.

```
void NvData_Example()
{
    NvDataTDrv *pNVDrv;
    UInt32     hDataSet1, hDataSet2;
    char       Zero[256];

    // Create driver object
    NvData_create(&pNVDrv);

    // Configure first data set 'Seat_Position' with an array of 4 doubles
    NvData_createDataSet(pNVDrv, &hDataSet1);
    NvData_setName(pNVDrv, hDataSet1, "Seat_Position");
    NvData_setType(pNVDrv, hDataSet1, NVDATA_TYPE_DOUBLE_FLOAT);
    NvData_setDimension(pNVDrv, hDataSet1, 4);

    // Configure second data set 'Error_History' with an array of 32 unsigned
    integers
    NvData_createDataSet(pNVDrv, &hDataSet2);
    NvData_setName(pNVDrv, hDataSet2, "Error_History");
    NvData_setType(pNVDrv, hDataSet2, NVDATA_TYPE_UINT32);
    NvData_setDimension(pNVDrv, hDataSet2, 32);

    // Apply and verify previously set configuration
    NvData_apply(pNVDrv);

    // Do an initial write of both data sets
    memset(Zero, 0, 256);
    NvData_write(pNVDrv, hDataSet1, Zero);
    NvData_write(pNVDrv, hDataSet2, Zero);
}
```

Serial Interface Communication

Introduction

This section contains the generic functions for communication via a serial interface.

The generic functions use a receive and transmit buffer to buffer the data. Because they do not have direct access to the UART, they are hardware-independent and can be used for different I/O boards. These generic functions are described in this chapter.

Where to go from here

Information in this section

Basic Principles of Serial Communication.....	412
Data Types for Serial Communication.....	417
Generic Serial Interface Communication Functions.....	424

Basic Principles of Serial Communication

Where to go from here

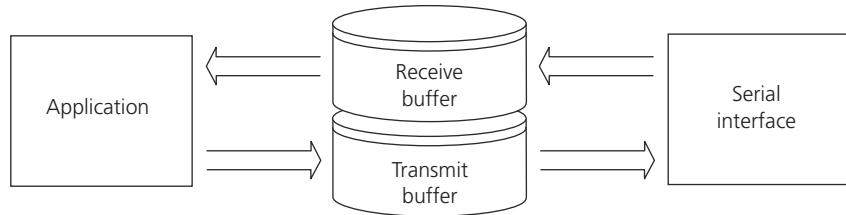
Information in this section

Software FIFO Buffer	412
To get information about the receive and transmit buffers.	
Trigger Levels	413
To get information about the trigger levels.	
How to Handle Subinterrupts in Serial Communication	413
Instructions on handling subinterrupts in serial communication.	
Example of a Serial Interface Communication	415
Shows you how to implement serial interface communication.	

Software FIFO Buffer

Introduction

The software FIFO buffer is a memory section that provides the UART with additional space for data storage and ensures that the generic functions are hardware-independent.



The software FIFO buffer stores data that will be written to (transmit buffer) or has been read by (receive buffer) the UART.

Transmit buffer

The transmit buffer is filled with data to be sent as long as free space is available. It cannot be overwritten. You can write data to the transmit buffer with the function `dsser_transmit`.

Receive buffer

The receive buffer is filled with data received by the UART as long as free space is available. If an overflow occurs, old data in the receive buffer is overwritten or new data is rejected. This depends on the mode of the FIFO. You can access the receive buffer by using the functions `dsser_receive` and `dsser_receive_term`.

Related topics**Basics**

[Trigger Levels](#)..... 413

References

dsse_receive	432
dsse_receive_term	433
dsse_transmit	430

Trigger Levels

Introduction

Two different trigger levels can be configured.

UART trigger level

The UART trigger level is hardware-dependent. After the specified number of bytes is received, the UART generates an interrupt and the bytes are copied into the receive buffer.

User trigger level

The user trigger level is hardware-independent and can be adjusted in smaller or larger steps than the UART trigger level. After a specified number of bytes is received in the receive buffer, the subinterrupt handler is called.

Related topics**Basics**

[Basic Principles of Serial Communication](#)..... 412

HowTos

[How to Handle Subinterrupts in Serial Communication](#)..... 413

How to Handle Subinterrupts in Serial Communication

Introduction

The interrupt functions must be used only in handcoded applications. Using them in Simulink applications (user code or S-functions) conflicts with the internal interrupt handling.

The following subinterrupts can be passed to your application:

Subinterrupt	Meaning
DSSER_TRIGGER_LEVEL_SUBINT	Generated when the receive buffer is filled with the number of bytes specified as the trigger level (see Trigger Levels on page 413).
DSSER_TX_FIFO_EMPTY_SUBINT	Generated when the transmit buffer has no data.
DSSER_RECEIVER_LINE_SUBINT	Line status interrupt provided by the UART.
DSSER_MODEM_STATE_SUBINT	Modem status interrupt provided by the UART.
DSSER_NO_SUBINT	Generated after the last subinterrupt. This subinterrupt tells your application that no further subinterrupts were generated.

Method

To install a subinterrupt handler within your application

- 1 Write a function that handles your subinterrupt, such as:

```
void my_subint_handler(dsserChannel* serCh, Int32 subint)
{
    switch (subint)
    {
        case DSSER_TRIGGER_LEVEL_SUBINT:
            /* do something */
            break;
        case DSSER_TX_FIFO_EMPTY_SUBINT:
            /* do something */
            break;
        case DSSER_NO_SUBINT:
            /* no further subinterrupts */
            break;
        default:
            break;
    }
}
```

- 2 Initialize your subinterrupt handler:

```
dsser_subint_handler_inst(serCh,
                           (dsser_subint_handler_t) my_subint_handler);
```

- 3 Enable the required subinterrupts:

```
dsser_subint_enable(serCh,
                     DSSE_TRIGGER_LEVEL_SUBINT_MASK |
                     DSSE_TX_FIFO_EMPTY_SUBINT_MASK);
```

Related topics

Basics

[Trigger Levels](#)..... 413

References

dsser_subint_enable	444
dsser_subint_handler_inst	443

dsser_subint_handler_t.....	421
dsserChannel.....	422

Example of a Serial Interface Communication

Example

The serial interface is initialized with 9600 baud, 8 data bits, 1 stop bit and no parity. The receiver FIFO generates a subinterrupt when it received 32 bytes and the subinterrupt handler `callback` is called. The subinterrupt handler `callback` reads the received bytes and sends the bytes back immediately.

```
#include <brtenv.h>
void callback(dsserChannel* serCh, UInt32 subint)
{
    UInt32 count;
    UInt8 data[32];
    switch (subint)
    {
        case DSSEER_TRIGGER_LEVEL_SUBINT:
            msg_info_set(0,0,"DSSEER_TRIGGER_LEVEL_SUBINT");
            dsser_receive(serCh,32,data,&count);
            dsser_transmit(serCh,count,data,&count);
            break;
        case DSSEER_TX_FIFO_EMPTY_SUBINT:
            msg_info_set(0,0,"DSSEER_TX_FIFO_EMPTY_SUBINT");
            break;
        default:
            break;
    }
}
main()
{
    dsserChannel* serCh;
    RTLlib_INIT();

    /* allocate a new 1024 byte SW-FIFO */
    serCh = dsser_init(DSSEER_ONBOARD, 0, 1024);
    dsser_subint_handler_inst(serCh,
        (dsser_subint_handler_t)callback);
}
```

```
dsser_subint_enable(serCh,
    DSSEr_TRIGGER_LEVEL_SUBINT_MASK |
    DSSEr_TX_FIFO_EMPTY_SUBINT_MASK);
/* config and start the UART */
dsseR_config(serCh, DSSEr_FIFO_MODE_OVERWRITE,
    9600, 8, DSSEr_1_STOPBIT, DSSEr_NO_PARITY,
    DSSEr_14_BYTE_TRIGGER_LEVEL, 32, DSSEr_RS232);
RTLIB_INT_ENABLE();
for(;;)
{
    RTLIB_BACKGROUND_SERVICE();
}
}
```

Data Types for Serial Communication

Introduction	There are some specific data structures specified for the serial communication interface.
---------------------	---

Where to go from here	Information in this section
dsser_ISR	417 Provides information about the interrupt identification register.
dsser_LSR	419 Provides information about the status of data transfers.
dsser_MSR	420 Provides information about the state of the control lines.
dsser_subint_handler_t	421 Provides information about the subinterrupt handler.
dsserChannel	422 Provides information about the serial channel.

dsser_ISR

Syntax

```
typedef union
{
    UInt32     Byte;
    struct
    {
        unsigned dummy : 24;
        unsigned DSSEr_FIFO_Status_Bit1 : 1;
        unsigned DSSEr_FIFO_Status_Bit0 : 1;
        unsigned DSSEr_Bit5 : 1;
        unsigned DSSEr_Bit4 : 1;
        unsigned DSSEr_Int_Priority_Bit2 : 1;
        unsigned DSSEr_Int_Priority_Bit1 : 1;
        unsigned DSSEr_Int_Priority_Bit0 : 1;
        unsigned DSSEr_Int_Status : 1;
    }Bit;
}dsser_ISR;
```

Include file

dsserdef.h

Description	The structure <code>dsser_ISR</code> provides information about the interrupt identification register (IIR). Call <code>dsser_status_read</code> to read the status register.
--------------------	---

Note

The data type contains the value of the UART's register. The register conforms to a standard 16550 UART such as the TEXAS INSTRUMENTS PC16550D. For further information, refer to <http://www.ti.com>.

Members

The structure provides the following members:

Member	Description
<code>DSSER_INT_STATUS</code>	0 if interrupt pending
<code>DSSER_INT_PRIORITY_BIT0</code>	Interrupt ID bit 1
<code>DSSER_INT_PRIORITY_BIT1</code>	Interrupt ID bit 2
<code>DSSER_INT_PRIORITY_BIT2</code>	Interrupt ID bit 3
<code>DSSER_BIT4</code>	Not relevant
<code>DSSER_BIT5</code>	Not relevant
<code>DSSER_FIFO_STATUS_BIT0</code>	UART FIFOs enabled
<code>DSSER_FIFO_STATUS_BIT1</code>	UART FIFOs enabled

For more information about the predefined constants, refer to the datasheet of the *TEXAS INSTRUMENTS, PC16550D*.

Related topics**References**

dsser_status_read	440
---	-----

dsser_LSR

Syntax

```
typedef union
{
    UInt32     Byte;
    struct
    {
        unsigned dummy : 24;
        unsigned DSser_FIFO_DATA_ERR : 1;
        unsigned DSser_THR_TSR_STATUS : 1;
        unsigned DSser_THR_STATUS : 1;
        unsigned DSser_BREAK_STATUS : 1;
        unsigned DSser_FRAMING_ERR : 1;
        unsigned DSser_PARITY_ERR : 1;
        unsigned DSser_OVERRUN_ERR : 1;
        unsigned DSser_RECEIVE_DATA_RDY : 1;
    } Bit;
} dsser_LSR;
```

Include file

dsserdef.h

Description

The structure `dsser_LSR` provides information about the status of data transfers. Call `dsser_status_read` to read the status register.

Note

The data type contains the value of the UART's register.
The register conforms to a standard 16550 UART such as the TEXAS INSTRUMENTS PC16550D. For further information, refer to <http://www.ti.com>.

Members

The structure provides the following members.

Member	Description
DSser_RECEIVE_DATA_RDY	Data ready (DR) indicator
DSser_OVERRUN_ERR	Overrun error (OE) indicator
DSser_PARITY_ERR	Parity error (PE) indicator
DSser_FRAMING_ERR	Framing error (FE) indicator
DSser_BREAK_STATUS	Break interrupt (BI) indicator
DSser_THR_STATUS	Transmitter holding register empty (THRE)
DSser_THR_TSR_STATUS	Transmitter empty (TEMT) indicator
DSser_FIFO_DATA_ERR	Error in receiver FIFO

For more information about the predefined constants, refer to the datasheet of the *TEXAS INSTRUMENTS, PC16550D*.

Related topics**References**

dsser_status_read	440
---	-----

dsser_MSR

Syntax

```
typedef union
{
    UInt32     Byte;
    struct
    {
        unsigned dummy : 24;
        unsigned DSSEr_OP2_STATUS : 1;
        unsigned DSSEr_OP1_STATUS : 1;
        unsigned DSSEr_DTR_STATUS : 1;
        unsigned DSSEr_RTS_STATUS : 1;
        unsigned DSSEr_CD_STATUS : 1;
        unsigned DSSEr_RI_STATUS : 1;
        unsigned DSSEr_DSR_STATUS : 1;
        unsigned DSSEr_CTS_STATUS : 1;
    }Bit;
}dsser_MSR;
```

Include file

dsserdef.h

Description

The structure `dsser_MSR` provides information about the state of the control lines. Call `dsser_status_read` to read the status register.

Note

The data type contains the value of the UART's register.

The register conforms to a standard 16550 UART such as the TEXAS INSTRUMENTS PC16550D. For further information, refer to <http://www.ti.com>.

Members

The structure provides the following members.

Member	Description
DSSER_CTS_STATUS	Clear-to-send (CTS) changed state
DSSER_DSR_STATUS	Data-set-ready (DSR) changed state
DSSER_RI_STATUS	Ring-indicator (RI) changed state
DSSER_CD_STATUS	Data-carrier-detect (CD) changed state
DSSER_RTS_STATUS	Complement of CTS
DSSER_DTR_STATUS	Complement of DSR
DSSER_OP1_STATUS	Complement of RI
DSSER_OP2_STATUS	Complement of DCD

For more information about the predefined constants, refer to the datasheet of the *TEXAS INSTRUMENTS, PC16550D*.

Related topics**References**

dsser_status_read	440
---	-----

dsser_subint_handler_t

Syntax

```
typedef void (*dsser_subint_handler_t) (void* serCh, Int32 subint)
```

Include file

`dsserdef.h`

Description

You must use this type definition if you install a subinterrupt handler (see [How to Handle Subinterrupts in Serial Communication](#) on page 413 or [dsser_subint_handler_inst](#) on page 443).

Members

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

subint Identification number of the related subinterrupt. The following symbols are predefined:

Predefined Symbol	Meaning
DSSEN_TRIGGER_LEVEL_SUBINT	Interrupt triggered when the user trigger level is reached (see Trigger Levels on page 413).

Predefined Symbol	Meaning
DSSER_TX_FIFO_EMPTY_SUBINT	Interrupt triggered when the transmit buffer is empty.
DSSER_RECEIVER_LINE_SUBINT	Line status interrupt of the UART.
DSSER_MODEM_STATE_SUBINT	Modem status interrupt of the UART.
DSSER_NO_SUBINT	Flag that is sent after the last triggered subinterrupt.

Related topics**Basics**

[Trigger Levels](#)..... 413

References

[dsser_init](#)..... 425

dsserChannel

Syntax

```
typedef struct
{
/*--- public -----*/
    /* interrupt status register */
    dsser_ISR intStatusReg;
    /* line status register */
    dsser_LSR lineStatusReg;
    /* modem status register */
    dsser_MSR modemStatusReg;
/*--- protected -----*/
    /*--- serial channel allocation ---*/
    UInt32 module;
    UInt32 channel;
    Int32 board_bt;
    UInt32 board;
    UInt32 fifo_size;
    UInt32 frequency;
```

```

/*--- serial channel configuration ---*/
UInt32 baudrate;
UInt32 databits;
UInt32 stopbits;
UInt32 parity;
UInt32 rs_mode;
UInt32 fifo_mode;
UInt32 uart_trigger_level;
UInt32 user_trigger_level;
dsser_subint_handler_t subint_handler;
dsserService* serService;
dsfifo_t* txFifo;
dsfifo_t* rxFifo;
UInt32 queue;
UInt8 isr;
UInt8 lsr;
UInt8 msr;
UInt32 interrupt_mode;
UInt8 subint_mask;
Int8 subint;
}dsserChannel

```

Include file	dsserdef.h
---------------------	------------

Description	This structure provides information about the serial channel. You can call dsser_status_read to read the values of the status registers. All protected variables are only for internal use.
--------------------	---

Members	intStatusReg Interrupt status register. Refer to dsser_ISR on page 417.
	lineStatusReg Line status register. Refer to dsser_LSR on page 419.
	modemStatusReg Modem status register. Refer to dsser_MSR on page 420.

Related topics	References
	dsser_status_read 440

Generic Serial Interface Communication Functions

Where to go from here

Information in this section

dsser_init	425
To initialize the serial interface and install the interrupt handler.	
dsser_free	426
To close a serial interface.	
dsser_config	427
To configure and start the serial interface.	
dsser_transmit	430
To transmit data through the serial interface.	
dsser_receive	432
To receive data through the serial interface.	
dsser_receive_term	433
To receive data through the serial interface.	
dsser_fifo_reset	435
To reset the serial interface.	
dsser_enable	436
To enable the serial interface.	
dsser_disable	436
To disable the serial interface.	
dsser_error_read	437
To read an error flag of the serial interface.	
dsser_transmit_fifo_level	438
To get the number of bytes in the transmit buffer.	
dsser_receive_fifo_level	439
To get the number of bytes in the receive buffer.	
dsser_status_read	440
To read the value of one or more status registers and store the values in the appropriate fields of the channel structure.	
dsser_handle_get	441
To check whether the serial interface is in use.	
dsser_set	442
To set a property of the UART.	
dsser_subint_handler_inst	443
To install a subinterrupt handler for the serial interface.	
dsser_subint_enable	444
To enable one or several subinterrupts of the serial interface.	

[dsser_subint_disable](#).....445

To disable one or several subinterrupts of the serial interface.

[dsser_word2bytes](#).....446

To convert a word (max. 4 bytes long) into a byte array.

[dsser_bytes2word](#).....448

To convert a byte array with a maximum of 4 elements into a single word.

dsser_init

Syntax

```
dsserChannel* dsser_init(
    UInt32 base,
    UInt32 channel,
    UInt32 fifo_size)
```

Include file

`dsser.h`

Purpose

To initialize the serial interface and install the interrupt handler.

Note

Pay attention to the initialization sequence. First, initialize the processor board, then the I/O boards, and then the serial interface.

Parameters

base Specifies the base address of the serial interface. This value has to be set to DSSER_ONBOARD.

channel Specifies the number of the channel to be used for the serial interface. The permitted value is 0.

fifo_size Specifies the size of the transmit and receive buffer in bytes. The size must be a power of two (2^n) and at least 64 bytes. The maximum size depends on the available memory.

Return value

This function returns the pointer to the serial channel structure.

Messages

The following messages are defined (x = base address of the I/O board, y = number of the channel):

ID	Type	Message	Description
100	Error	x, ch=y, Board not found!	I/O board was not found.
101	Warning	x, ch=y, Mixed usage of high and low level API!	It is not allowed to use the generic functions (high-level access functions) and the low-level access functions of the serial interface on the same channel. It is recommended to use only the generic functions.
501	Error	x, ch=y, memory: Allocation error on master.	Memory allocation error. No free memory on the master.
508	Error	x, ch=y, channel: out of range!	The channel parameter is out of range.
700	Error	x, ch=y, Buffersize: Illegal	The fifo_size parameter is out of range.

Related topics**Basics**

[Basic Principles of Serial Communication](#)..... 412

Examples

[Example of a Serial Interface Communication](#)..... 415

References

[Data Types for Serial Communication](#)..... 417
[dsser_config](#)..... 427
[dsser_free](#)..... 426

dsser_free

Syntax

```
Int32 dsser_free(dsserChannel*serCh)
```

Include file

dsser.h

Purpose

To close a serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation. The specified serial interface is closed. Its memory for the buffer is freed and the interrupts are released. A serial interface can be created again using the <code>dsser_init</code> function.
DSSER_TX_FIFO_NOT_EMPTY	The serial interface is not closed, because the transmit buffer is not empty.
DSSER_CHANNEL_INIT_ERROR	There is no serial interface to be closed (<code>serCh == NULL</code>).

Related topics

Basics

[Basic Principles of Serial Communication](#)..... 412

References

[dsser_init](#)..... 425

dsser_config

Syntax

```
void dsser_config(
    dsserChannel* serCh,
    const UInt32 fifo_mode,
    const UInt32 baudrate,
    const UInt32 databits,
    const UInt32 stopbits,
    const UInt32 parity,
    const UInt32 uart_trigger_level,
    const Int32 user_trigger_level,
    const UInt32 uart_mode)
```

Include file

`dsser.h`

Purpose

To configure and start the serial interface.

Note

- This function starts the serial interface. Therefore, all dSPACE real-time boards must be initialized and the interrupt vector must be installed before calling this function.
- Calling this function again reconfigures the serial interface.

Parameters **serCh** Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

fifo_mode Specifies the mode of the receive buffer (see [Software FIFO Buffer](#) on page 412):

Value	Mode	Meaning
DSSER_FIFO_MODE_BLOCKED	Blocked mode	If the receive buffer is full, new data is rejected.
DSSER_FIFO_MODE_OVERWRITE	Overwrite mode	If the receive buffer is full, new data replaces the oldest data in the buffer.

baudrate Specifies the baud rate in bits per second:

Mode	Baud Rate Range
RS232	5 ... 230,400 baud
RS422	5 ... 10,000,000 baud
RS485	5 ... 10,000,000 baud

For further information, refer to [Serial Interface \(MicroLabBox Features\)](#).

databits Specifies the number of data bits. Values are: 5, 6, 7, 8.

stopbits Specifies the number of stop bits. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_1_STOPBIT	1 stop bit
DSSER_2_STOPBIT	The number of stop bits depends on the number of the specified data bits: 5 data bits: 1.5 stop bits 6 data bits: 2 stop bits 7 data bits: 2 stop bits 8 data bits: 2 stop bits

parity Specifies whether and how parity bits are generated. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_PARITY	No parity bits
DSSER_ODD_PARITY	Parity bit is set so that there is an odd number of "1" bits in the byte, including the parity bit.
DSSER_EVEN_PARITY	Parity bit is set so that there is an even number of "1" bits in the byte, including the parity bit.
DSSER_FORCED_PARITY_ONE	Parity bit is forced to a logic 1.
DSSER_FORCED_PARITY_ZERO	Parity bit is forced to a logic 0.

uart_trigger_level Sets the UART trigger level (see [Trigger Levels](#) on page 413). The following symbols are predefined:

Predefined Symbol	Meaning
DSER_1_BYTE_TRIGGER_LEVEL	1-byte trigger level
DSER_4_BYTE_TRIGGER_LEVEL	4-byte trigger level
DSER_8_BYTE_TRIGGER_LEVEL	8-byte trigger level
DSER_14_BYTE_TRIGGER_LEVEL	14-byte trigger level

Note

Use the highest UART trigger level possible to generate fewer interrupts.

user_trigger_level Sets the user trigger level within the range of 1 ... (fifo_size - 1) for the receive interrupt (see [Trigger Levels](#) on page 413):

Value	Meaning
DSER_DEFAULT_TRIGGER_LEVEL	Synchronizes the UART trigger level and the user trigger level.
1 ... (fifo_size - 1)	Sets the user trigger level.
DSER_TRIGGER_LEVEL_DISABLE	No receive subinterrupt handling for the serial interface

uart_mode Sets the mode of the UART transceiver.

The following symbols are predefined:

Predefined Symbol	Meaning
DSER_RS232	RS232 mode
DSER_AUTOFLOW_DISABLE	Transfer without HW handshake (RTS/CTS)
DSER_AUTOFLOW_ENABLE	Transfer with HW handshake (RTS/CTS)

Messages

The following messages are defined (x = base address of the I/O board, y = number of the channel):

ID	Type	Message	Description
101	Warning	x, ch=y, Mixed usage of high and low level API!	It is not allowed to use the generic functions (high-level access functions) and the low-level access functions of the serial interface on the same channel. It is recommended to use only the generic functions.
601	Error	x, serCh: The UART channel was not initialized.	The <code>dsser_config</code> function was called before the serial interface was initialized with <code>dsser_init</code> .
602	Error	x, ch=y, baudrate: Illegal!	The <code>baudrate</code> parameter is out of range.
603	Error	x, ch=y, databits: Use range 5 ... 8 bits!	The <code>databits</code> parameter is out of range.

ID	Type	Message	Description
604	Error	x, ch=y, stopbits: Illegal number (1-2 bits allowed)!	The <code>stopbits</code> parameter is out of range.
605	Error	x, ch=y, parity: Illegal parity!	The <code>parity</code> parameter is out of range.
606	Error	x, ch=y, trigger_level: Illegal UART trigger level!	The <code>uart_trigger_level</code> parameter is out of range.
607	Error	x, ch=y, trigger_level: Illegal user trigger level!	The <code>user_trigger_level</code> parameter is out of range.
608	Error	x, ch=y, fifo_mode: Use range 0 ... (fifo_size-1) bytes!	The <code>uart_mode</code> parameter is out of range.
609	Error	x, ch=y, uart_mode: Transceiver not supported!	The selected UART mode does not exist for this serial interface.
611	Error	x, ch=y, uart_mode: Autoflow is not supported!	Autoflow does not exist for this serial interface.

Related topics**Basics**

[Basic Principles of Serial Communication](#)..... 412

Examples

[Example of a Serial Interface Communication](#)..... 415

References

[dsser_init](#)..... 425

dsser_transmit

Syntax

```
Int32 dsser_transmit(
    dsserChannel* serCh,
    UInt32 dataLen,
    UInt8* data,
    UInt32* count)
```

Include file**dsser.h****Purpose**

To transmit data through the serial interface.

Parameters	<p>serCh Specifies the pointer to the serial channel structure (see dsser_init on page 425).</p> <p>datalen Specifies the number of bytes to be transmitted.</p> <p>data Specifies the pointer to the data to be transmitted.</p> <p>count Specifies the pointer to the number of transmitted bytes. When this function is finished, the variable contains the number of bytes that were transmitted. If the function was able to send all the data, the value is equal to the value of the datalen parameter.</p>
-------------------	---

Return value	This function returns an error code. The following symbols are predefined:
---------------------	--

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_FIFO_OVERFLOW	The FIFO is filled or not all the data could be copied to the FIFO.
DSSER_COMMUNICATION_FAILED	The function failed with no effect on the input or output data. No data is written to the FIFO. The communication between the real-time processor and the UART is might be overloaded. Do not poll this function because it may cause an endless loop.

Example	This example shows how to check the transmit buffer for sufficient free memory before transmitting data.
----------------	--

```
UInt32 count;
UInt8 block[5] = {1, 2, 3, 4, 5};
if(dsser_transmit_fifo_level(serCh) < serCh->fifo_size - 5)
{
    dsser_transmit(serCh, 5, block, &count);
}
```

Related topics	<p>Basics</p> <p>Basic Principles of Serial Communication..... 412</p> <p>Examples</p> <p>Example of a Serial Interface Communication..... 415</p> <p>References</p> <p>dsser_init..... 425</p> <p>dsser_transmit_fifo_level..... 438</p>
-----------------------	--

dsser_receive

Syntax

```
Int32 dsser_receive(
    dsserChannel1* serCh,
    UInt32 dataLen,
    UInt8* data,
    UInt32* count)
```

Include file

`dsser.h`

Purpose

To receive data through the serial interface.

Tip

It is better to receive a block of bytes instead of several single bytes because the processing speed is faster.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

dataLen Specifies the number of data bytes to be read. The value must not be greater than the FIFO size defined with [dsser_init](#).

data Specifies the pointer to the destination buffer.

count Specifies the pointer to the number of received bytes. When this function is finished, the variable contains the number of bytes that were received.

Return value

This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
<code>DSSER_NO_ERROR</code>	No error occurred during the operation.
<code>DSSER_NO_DATA</code>	No new data is read from the FIFO.
<code>DSSER_FIFO_OVERFLOW</code>	The FIFO is filled. The behavior depends on the <code>fifo_mode</code> adjusted with dsser_config : <ul style="list-style-type: none"> ▪ <code>fifo_mode = DSSER_FIFO_MODE_BLOCKED</code> Not all new data could be placed in the FIFO. ▪ <code>fifo_mode = DSSER_FIFO_MODE_OVERWRITE</code> The old data is rejected.
<code>DSSER_COMMUNICATION_FAILED</code>	The function failed with no effect on the input or output data. No data is read from the FIFO. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Example

The following example shows how to receive 4 bytes.

```
UInt8 data[4];
UInt32 count;
Int32 error;
/* receive four bytes over serCh */
error = dsser_receive(serCh, 4, data, &count);
```

Related topics**Basics**

Basic Principles of Serial Communication.....	412
---	-----

Examples

Example of a Serial Interface Communication.....	415
--	-----

References

dsser_init.....	425
-----------------	-----

dsser_receive_term

Syntax

```
Int32 dsser_receive_term(
    dsserChannel* serCh,
    UInt32 datalen,
    UInt8* data,
    UInt32* count,
    const UInt8 term)
```

Include file

dsser.h

Purpose

To receive data through the serial interface.

Description

This function is terminated when the character **term** is received. The character **term** is stored as the last character in the buffer, so you can check if the function was completed.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

dataLEN Specifies the number of data bytes to be read. The value must not be greater than the FIFO size defined with `dsser_init`.

data Specifies the pointer to the destination buffer.

count Specifies the pointer to the number of received bytes. When this function is finished, the variable contains the number of bytes that were received.

term Specifies the character that terminates the reception of bytes.

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_NO_DATA	No new data is read from the FIFO.
DSSER_FIFO_OVERFLOW	The FIFO is filled. The behavior depends on the <code>fifo_mode</code> adjusted with <code>dsser_config</code> : <ul style="list-style-type: none"> ▪ <code>fifo_mode = DSSER_FIFO_MODE_BLOCKED</code> Not all new data could be placed in the FIFO. ▪ <code>fifo_mode = DSSER_FIFO_MODE_OVERWRITE</code> The old data is rejected.
DSSER_COMMUNICATION_FAILED	The function failed with no effect on the input or output data. No data is read from the FIFO. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Example

The following example shows how to receive a maximum of 4 bytes via the serial channel until the terminating character '\r' occurs:

```
UInt8 data[4];
UInt32 count;
Int32 error;
error = dsser_receive_term(serCh, 4, data, &count, '\r');
```

Related topics

Basics

Basic Principles of Serial Communication.....	412
---	-----

References

dsser_init.....	425
-----------------	-----

dsser_fifo_reset

Syntax

```
Int32 dsser_fifo_reset(dsserChannel* serCh)
```

Include file

`dsser.h`

Purpose

To reset the serial interface.

Description

The channel is disabled and the transmit and receive buffers are cleared.

Note

If you want to continue to use the serial interface, the channel has to be enabled with `dsser_enable`.

Parameters

serCh Specifies the pointer to the serial channel structure (see `dsser_init` on page 425).

Return value

This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
<code>DSSER_NO_ERROR</code>	No error occurred during the operation.
<code>DSSER_COMMUNICATION_FAILED</code>	The function failed. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics

Basics

Basic Principles of Serial Communication	412
--	-----

References

dsser_enable	436
dsser_init	425

dsser_enable

Syntax

```
Int32 dsser_enable(const dsserChannel* serCh)
```

Include file

`dsser.h`

Purpose

To enable the serial interface.

Description

The UART interrupt is enabled, the serial interface starts transmitting and receiving data.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

Return value

This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
<code>DSSER_NO_ERROR</code>	No error occurred during the operation.
<code>DSSER_COMMUNICATION_FAILED</code>	The function failed. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics

Basics

[Basic Principles of Serial Communication](#)..... 412

References

dsser_disable	436
dsser_init	425

dsser_disable

Syntax

```
Int32 dsser_disable(const dsserChannel* serCh)
```

Include file	<code>dsser.h</code>						
Purpose	To disable the serial interface.						
Description	The serial interface stops transmitting data, incoming data is no longer stored in the receive buffer and the UART subinterrupts are disabled.						
Parameters	serCh Specifies the pointer to the serial channel structure (see dsser_init on page 425).						
Return value	This function returns an error code. The following symbols are predefined:						
Predefined Symbol	Meaning						
<code>DSSER_NO_ERROR</code>	No error occurred during the operation.						
<code>DSSER_COMMUNICATION_FAILED</code>	The function failed. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.						
Related topics	<p>Basics</p> <table border="0"> <tr> <td style="background-color: #e0e0e0;">Basic Principles of Serial Communication.....</td> <td style="background-color: #e0e0e0;">412</td> </tr> </table> <p>References</p> <table border="0"> <tr> <td style="background-color: #e0e0e0;">dsser_enable.....</td> <td style="background-color: #e0e0e0;">436</td> </tr> <tr> <td style="background-color: #e0e0e0;">dsser_init.....</td> <td style="background-color: #e0e0e0;">425</td> </tr> </table>	Basic Principles of Serial Communication	412	dsser_enable	436	dsser_init	425
Basic Principles of Serial Communication	412						
dsser_enable	436						
dsser_init	425						

[dsser_error_read](#)

Syntax	<code>Int32 dsser_error_read(const dsserChannel* serCh)</code>
Include file	<code>dsser.h</code>
Purpose	To read an error flag of the serial interface.

Description	Because only one error flag is returned, you have to call this function as long as the value <code>DSSER_NO_ERROR</code> is returned to get all error flags.
Parameters	serCh Specifies the pointer to the serial channel structure (see dsser_init on page 425).
Return value	This function returns an error flag.
Related topics	Basics Basic Principles of Serial Communication 412 References dsser_config 427 dsser_init 425

dsser_transmit_fifo_level

Syntax	<code>Int32 dsser_transmit_fifo_level(const dsserChannel* serCh)</code>
Include file	<code>dsser.h</code>
Purpose	To get the number of bytes in the transmit buffer.
Parameters	serCh Specifies the pointer to the serial channel structure (see dsser_init on page 425).
Return value	This function returns the number of bytes in the transmit buffer.

Related topics

Basics

[Basic Principles of Serial Communication](#)..... 412

References

[dsser_init](#)..... 425

[dsser_receive_fifo_level](#)..... 439

dsser_receive_fifo_level

Syntax

```
Int32 dsser_receive_fifo_level(const dsserChannel* serCh)
```

Include file`dsser.h`**Purpose**

To get the number of bytes in the receive buffer.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

Return value

This function returns the number of bytes in the receive buffer.

Related topics

Basics

[Basic Principles of Serial Communication](#)..... 412

References

[dsser_init](#)..... 425

[dsser_transmit_fifo_level](#)..... 438

dsser_status_read

Syntax

```
Int32 dsser_status_read(
    dsserChannel*serCh,
    const UInt8 register_type)
```

Include file

`dsser.h`

Purpose

To read the value of one or more status registers and to store the values in the appropriate fields of the channel structure.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

register_type Specifies the register that is read. You can combine the predefined symbols with the logical operator OR to read several registers. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_STATUS_IIR_FCR	Interrupt status register, see dsser_ISR data type.
DSSER_STATUS_LSR	Line status register, see dsser_ISR data type.
DSSER_STATUS_MSR	Modem status register, see dsser_ISR data type.

Return value

This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Example

This example shows how to check if the clear-to-send bit has changed:

```
UInt8 cts;
dsser_status_read(serCh, DSSER_STATUS_MSR);
cts = serCh->modemStatusReg.Bit.DSSER_CTS_STATUS;
```

Related topics**Basics**

Basic Principles of Serial Communication	412
--	-----

References

dsser_init	425
dsser_ISR	417
dsser_LSR	419
dsser_MSR	420

dsser_handle_get

Syntax

```
dsserChannel* dsser_handle_get(
    UInt32 base,
    UInt32 channel)
```

Include file`dsser.h`**Purpose**

To check whether the serial interface is in use.

Parameters

base Specifies the base address of the serial interface. This value has to be set to DSSER_ONBOARD.

channel Specifies the number of the channel to be used for the serial interface. The permitted value is 0.

Return value

This function returns:

- NULL if the specified serial interface is not used.
- A pointer to the serial channel structure of the serial interface that has been created by using the `dsser_init` function.

Related topics**Basics**

Basic Principles of Serial Communication	412
--	-----

References

dsser_init	425
----------------------------------	-----

dsser_set

Syntax

```
Int32 dsser_set(
    dsserChannel *serCh,
    UInt32 type,
    const void *value_p)
```

Include file

`dsser.h`

Purpose

To set a property of the UART.

Description

MicroLabBox is delivered with a standard quartz working with the frequency of $1.8432 \cdot 10^6$ Hz. You can replace this quartz with another one with a different frequency. Then you have to set the new quartz frequency using `dsser_set` followed by executing `dsser_config`.

Note

You must execute `dsser_config` after `dsser_set`; otherwise `dsser_set` has no effect.

Parameters

serCh Specifies the pointer to the serial channel structure (see `dsser_init` on page 425).

type Specifies the property to be changed (`DSSER_SET_UART_FREQUENCY`).

value_p Specifies the pointer to a `UInt32`-variable with the new value, for example, a variable which contains the quartz frequency.

Return value

This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
<code>DSSER_NO_ERROR</code>	No error occurred during the operation.
<code>DSSER_COMMUNICATION_FAILED</code>	The function failed. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Example

This example sets a new value for the frequency.

```
UInt32 freq = 1843200; /* 1.8432 MHz */
Int32 error;
error = dsser_set(serCh, DSSER_SET_UART_FREQUENCY, &freq);
```

Related topics**Basics**

Basic Principles of Serial Communication	412
--	-----

References

dsser_config	427
dsser_init	425

dsser_subint_handler_inst

Syntax

```
dsser_subint_handler_t dsser_subint_handler_inst(
    dsserChannel1* serCh,
    dsser_subint_handler_t subint_handler)
```

Include file

dsser.h

Purpose

To install a subinterrupt handler for the serial interface.

DescriptionAfter installing the handler, the specified subinterrupt type must be enabled (see [dsser_subint_enable](#) on page 444).**Note**

The interrupt functions must be used only in handcoded applications. Using them in Simulink applications (user code or S-functions) conflicts with the internal interrupt handling.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

subint_handler Specifies the pointer to the subinterrupt handler.

Return value

This function returns the pointer to the previously installed subinterrupt handler.

Related topics**Basics**

[Basic Principles of Serial Communication](#)..... 412

Examples

[Example of a Serial Interface Communication](#)..... 415

References

[dsser_init](#)..... 425

[dsser_subint_disable](#)..... 445

[dsser_subint_enable](#)..... 444

dsser_subint_enable

Syntax

```
Int32 dsser_subint_enable(
    dsserChannel* serCh,
    const UInt8 subint)
```

Include file**dsser.h****Purpose**

To enable one or several subinterrupts of the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

subint Specifies the subinterrupts to be enabled. You can combine the predefined symbols with the logical operator OR to enable several subinterrupts. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_TRIGGER_LEVEL_SUBINT_MASK	Interrupt triggered when the user trigger level is reached (see Trigger Levels on page 413)
DSSER_TX_FIFO_EMPTY_SUBINT_MASK	Interrupt triggered when the transmit buffer is empty

Predefined Symbol	Meaning
DSSER_RECEIVER_LINE_SUBINT_MASK	Line status interrupt of the UART
DSSER_MODEM_STATE_SUBINT_MASK	Modem status interrupt of the UART

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics Basics

[Basic Principles of Serial Communication](#)..... 412

Examples

[Example of a Serial Interface Communication](#)..... 415

References

[dsser_init](#)..... 425

[dsser_subint_disable](#)..... 445

[dsser_subint_handler_inst](#)..... 443

dsser_subint_disable

Syntax

```
Int32 dsser_subint_disable(
    dsserChannel* serCh,
    const UInt8 subint)
```

Include file

`dsser.h`

Purpose

To disable one or several subinterrupts of the serial interface.

Parameters

serCh Specifies the pointer to the serial channel structure (see [dsser_init](#) on page 425).

subint Specifies the subinterrupts to be disabled. You can combine the predefined symbols with the logical operator OR to disable several subinterrupts. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_TRIGGER_LEVEL_SUBINT_MASK	Interrupt triggered when the user trigger level is reached (see Trigger Levels on page 413)
DSSER_TX_FIFO_EMPTY_SUBINT_MASK	Interrupt triggered when the transmit buffer is empty
DSSER_RECEIVER_LINE_SUBINT_MASK	Line status interrupt of the UART
DSSER_MODEM_STATE_SUBINT_MASK	Modem status interrupt of the UART

Return value This function returns an error code. The following symbols are predefined:

Predefined Symbol	Meaning
DSSER_NO_ERROR	No error occurred during the operation.
DSSER_COMMUNICATION_FAILED	The function failed. The communication between the real-time processor and the UART might be overloaded. Do not poll this function because it might cause an endless loop.

Related topics

Basics

Basic Principles of Serial Communication.....	412
---	-----

References

dsser_init.....	425
dsser_subint_enable.....	444
dsser_subint_handler_inst.....	443

dsser_word2bytes

Syntax

```
UInt8* dsser_word2bytes(
    const UInt32* word,
    UInt8* bytes,
    const int bytesInWord)
```

Include file

dsser.h

Purpose

To convert a word (max. 4 bytes long) into a byte array.

Parameters	<p>word Specifies the pointer to the input word.</p> <p>bytes Specifies the pointer to the byte array. The byte array must have enough memory for bytesInWord elements.</p> <p>bytesInWord Specifies the number of elements in the byte array. Possible values are 2, 3, 4.</p>
Return value	This function returns the pointer to a byte array.
Example	<p>The following example shows how to write a processor-independent function that transmits a 32-bit value:</p> <pre>void word_transmit(dsserChannel* serCh, UInt32* word, UInt32* count) { UInt8 bytes[4]; UInt8* data_p; if(dsser_transmit_fifo_level(serCh) < serCh->fifo_size - 4) { data_p = dsser_word2bytes(word, bytes, 4); dsser_transmit(serCh, 4, data_p, count); } else { *count = 0; } }</pre>

Use of the function:

```
UInt32 word = 0x12345678;
UInt32 count;
word_transmit(serCh, &word, &count);
```

Related topics	<p>Basics</p> <table border="0"> <tr> <td style="vertical-align: top;"> Basic Principles of Serial Communication.....</td><td style="vertical-align: top;">412</td></tr> </table> <p>References</p> <table border="0"> <tr> <td style="vertical-align: top;"> dsser_bytes2word.....</td><td style="vertical-align: top;">448</td></tr> <tr> <td style="vertical-align: top;"> dsser_transmit.....</td><td style="vertical-align: top;">430</td></tr> <tr> <td style="vertical-align: top;"> dsser_transmit_fifo_level.....</td><td style="vertical-align: top;">438</td></tr> </table>	Basic Principles of Serial Communication	412	dsser_bytes2word	448	dsser_transmit	430	dsser_transmit_fifo_level	438
Basic Principles of Serial Communication	412								
dsser_bytes2word	448								
dsser_transmit	430								
dsser_transmit_fifo_level	438								

dsser_bytes2word

Syntax

```
UInt32* dsser_bytes2word(
    UInt8* bytes_p,
    UInt32* word_p,
    const int bytesInWord)
```

Include file

`dsser.h`

Purpose

To convert a byte array with a maximum of 4 elements into a single word.

Parameters

bytes_p Specifies the pointer to the input byte array.
word_p Specifies the pointer to the converted word.
bytesInWord Specifies the number of elements in the byte array. Possible values are 2, 3, 4.

Return value

This function returns the pointer to the converted word.

Example

The following example shows how to write a processor-independent function that receives a 32-bit value:

```
void word_receive(dsserChannel* serCh, UInt32* word_p, UInt32* count)
{
    UInt8 bytes[4];
    if(dsser_receive_fifo_level(serCh) > 3)
    {
        dsser_receive(serCh, 4, bytes, count);
        word_p = dsser_bytes2word(bytes, word_p, 4);
    }
    else
    {
        *count = 0;
    }
}
```

Use of the function:

```
UInt32 word;
UInt32 count;
word_receive(serCh, &word, &count);
```

Related topics**Basics**

[Basic Principles of Serial Communication.....](#) 412

References

[dsser_receive.....](#) 432
[dsser_receive_fifo_level.....](#) 439
[dsser_word2bytes.....](#) 446

Serial Peripheral Interface (SPI)

Introduction

The board's RTLib provides functions to implement synchronous communication with external devices using a serial peripheral interface.

For further information, refer to [Serial Peripheral Interface \(MicroLabBox Features\)](#).

Where to go from here

Information in this section

DioCl1Spi_apply	453
To apply the initialization settings to the SPI master based on the DIO Class 1 unit.	
DioCl1Spi_create	454
To create the I/O driver object for a serial peripheral interface master via DIO class 1 unit.	
DioCl1Spi_disableInterrupts	456
To disable the generation of interrupts from the SPI master based on the DIO Class 1 unit.	
DioCl1Spi_enableInterrupts	457
To enable the generation of interrupts from the SPI master based on the DIO Class 1 unit.	
DioCl1Spi_setChipSelectPolarity	459
To set the polarity of the chip select channels of the specified SPI unit.	
DioCl1Spi_setInterruptMode	460
To specify the interrupt mode of the current SPI unit.	
DioCl1Spi_setIoIntVector	461
To specify the interrupt handler processing interrupts from the SPI unit.	
DioCl1Spi_setOutputsHighZ	462
To set the state of the SPI output channels on the DIO Class 1 unit to high impedance.	

DioCl1Spi_setSignalVoltage	463
To set the voltage level of the SPI output signals.	
DioCl1Spi_setTriggerMode	464
To specify the trigger mode of the current SPI unit.	
DioCl1Spi_setTriggerLineOut	466
To specify the trigger line used for trigger signals of the SPI unit.	
DioCl1Spi_setTriggerLineOutStatus	467
To set the status of the trigger line.	
DioCl1Spi_start	468
To activate the specified SPI master based on the channels of the DIO Class 1 unit.	
DioCl1Spi_write	469
To write data to the channels of the current SPI unit during run time of the real-time application.	
DioCl1SpiCycle_apply	470
To apply the initialization settings to the specified SPI cycle.	
DioCl1SpiCycle_create	471
To create the I/O driver object for an SPI cycle configuration.	
DioCl1SpiCycle_getCycleData	473
To get the SPI cycle data received by the SPI unit.	
DioCl1SpiCycle_read	475
To read cycle data from the input channels of the current SPI unit.	
DioCl1SpiCycle_setBaudRate	476
To set the baud rate used for transferring an SPI cycle.	
DioCl1SpiCycle_setBitDirection	477
To set the bit order of an SPI cycle when transferring data.	
DioCl1SpiCycle_setChipSelectConf	478
To set the chip select configuration of an SPI cycle.	
DioCl1SpiCycle_setClockPolarity	479
To set the polarity of the clock signal.	
DioCl1SpiCycle_setClockPhase	480
To set the phase of the clock signal.	
DioCl1SpiCycle_setCycleData	482
To set the SPI cycle data to be transferred.	
DioCl1SpiCycle_setTimeAfterTran	483
To set the time after transfer value of an SPI cycle.	
DioCl1SpiCycle_setTimeBeforeTran	484
To set the time before transfer value of an SPI cycle.	
DioCl1SpiCycle_setTimeBetwWords	486
To set the time between words value for an SPI cycle.	

[DioCl1SpiCycle_setTimeCslnactive](#).....487
To set the minimum inactive time of the chip select signal.

[DioCl1SpiCycle_start](#).....488
To enable the transfer of SPI cycle data.

[DioCl1SpiCycle_write](#).....489
To transfer the cycle data to the output channels of the current SPI unit.

DioCl1Spi_apply

Syntax

```
UInt32 DioCl1Spi_apply(DioCl1SpiSDrvObject *pDioCl1Spi)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To apply the initialization settings to the SPI master based on the DIO Class 1 unit.

Description

Before you can start the specified DIO Class 1 unit, you have to transfer the settings of the DioCl1Spi driver object to its related input and output channels by using `DioCl1Spi_apply`. You have to do this also for the default values and for values that you specified by using the `DioCl1Spi_set<Parameter>` functions. The `DioCl1Spi_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The SPI communication on the specified digital input and output channels is not activated until you have called `DioCl1Spi_start`. Otherwise, the digital outputs of the DIO Class 1 unit are in a high impedance state.

To update settings during run time, you have to use `DioCl1Spi_write`.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters **pDioCl1Spi** Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using **DioCl1Spi_create**.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1Spi_create

Syntax

```
UInt32 DioCl1Spi_create(
    DioCl1SpiDrvObject **ppDioCl1Spi,
    UInt32 SpiUnitNo,
    UInt32 InputPort,
    UInt32 InputChannel,
    UInt32 OutputPort,
    UInt32 OutputChannel,
    UInt32 ChipSelectCount)
```

Include file

IoDrvDioClass1Spi.h

Purpose

To create the I/O driver object for a serial peripheral interface master via the DIO class 1 unit.

Description

This function performs all the steps necessary to create a specific I/O driver object for providing the master functionality of a serial peripheral interface (SPI) unit.

You can specify up to four SPI units. An SPI unit consists of one input channel (MISO) and at least three output channels (CLK, MOSI, CS1). Up to three additional chip select channels (CS2, CS3, CS4) can be configured. The output channels must belong to the same port.

This function also sets initial values so that the driver is ready for use, except for the cycle configuration (refer to **DioCl1SpiCycle_create**.)

The initial values are:

- Output signal voltage level: 2.5 V
- Polarity of the chip select signal: low active
- Interrupt generation: disabled
- Trigger generation: disabled

To completely set up a communication using the SPI master, you have to create a driver object for an SPI cycle with the same SPI unit number.

The function is intended to be called during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

ppDioCl1Spi Lets you specify the address of a variable which provides the address of the created SPI driver object.

SpiUnitNo Lets you specify the number of the SPI unit to be used in the range 1 ... 4.

Symbol	Meaning
DIO_CLASS1_SPI_UNIT_1	Specifies SPI unit 1.
DIO_CLASS1_SPI_UNIT_2	Specifies SPI unit 2.
DIO_CLASS1_SPI_UNIT_3	Specifies SPI unit 3.
DIO_CLASS1_SPI_UNIT_4	Specifies SPI unit 4.

InputPort Lets you specify the number of the DIO class 1 port that contains the input channel of the specified SPI master in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
DIO_CLASS1_PORT_2	Specifies port 2 of the DIO Class 1 unit.
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

InputChannel Lets you specify the channel to be used for the MISO signal of the specified SPI unit in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the currently used port of the DIO Class 1 unit.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the currently used port of the DIO Class 1 unit.

OutputPort Lets you specify the number of the DIO class 1 port that contains the output channels of the specified SPI master in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
DIO_CLASS1_PORT_2	Specifies port 2 of the DIO Class 1 unit.
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

OutputChannel Lets you specify the first output channel (CLK) of the specified SPI unit by using the predefined symbols DIO_CLASS1_CHANNEL_n with n in the range 1 ... 15 - **ChipSelectCount**.

At least two further channels are automatically allocated to provide channels for the MOSI and CS1 signals. Depending on the specified number of chip select channels (refer to **ChipSelectCount** parameter), up to three channels are additionally allocated for the CS2, CS3, and CS4 signals.

ChipSelectCount Lets you specify the number of chip select channels in the range 1 ... 4.

Symbol	Meaning
DIO_CLASS1_SPI_CS_COUNT_1	Specifies one chip select channel to be used (CS1).
...	...
DIO_CLASS1_SPI_CS_COUNT_4	Specifies four chip select channels to be used (CS1 ... CS4).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1Spi_disableInterrupts

Syntax

```
UInt32 DioCl1Spi_disableInterrupts(
    DioCl1SpiSDrvObject *pDioCl1Spi,
    UInt32 InterruptSelect)
```

Include file

IoDrvDioClass1Spi.h

Purpose	To disable the generation of interrupts from the SPI master based on the DIO Class 1 unit.						
Description	<p>With this function, you can disable the generation of interrupts that you specified before by using DioCl1Spi_setInterruptMode. The settings take effect immediately.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Serial Peripheral Interface (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using DioCl1Spi_create.</p> <p>InterruptSelect Lets you specify the interrupts that are to be disabled. You can only specify the <i>end of cycle</i> interrupt.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_INT_END_OF_CYCLE</td><td>The generation of the <i>end of cycle</i> interrupts of the current SPI unit will be disabled.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_INT_END_OF_CYCLE	The generation of the <i>end of cycle</i> interrupts of the current SPI unit will be disabled.		
Symbol	Meaning						
DIO_CLASS1_INT_END_OF_CYCLE	The generation of the <i>end of cycle</i> interrupts of the current SPI unit will be disabled.						
Return value	The function returns an error code.						
Syntax	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1Spi_enableInterrupts

Syntax	<pre>UInt32 DioCl1Spi_enableInterrupts(DioCl1SpiSDrvObject *pDioCl1Spi, UInt32 InterruptSelect)</pre>
---------------	--

Include file	<code>IoDrvDioClass1Spi.h</code>						
Purpose	To enable the generation of interrupts from the SPI master based on the DIO Class 1 unit.						
Description	<p>With this function, you can either enable the generation of interrupts or reenable the generation of interrupts that you disabled before by using <code>DioCl1Spi_disableInterrupts</code>. The interrupts have to be specified before by using <code>DioCl1Spi_setInterruptMode</code>. The settings take effect immediately.</p> <p>After calling <code>DioCl1Spi_start</code>, interrupts are always disabled.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Serial Peripheral Interface (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using <code>DioCl1Spi_create</code>.</p> <p>InterruptSelect Lets you specify the interrupts that are to be enabled. You can only specify the <i>end of cycle</i> interrupt.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>DIO_CLASS1_INT_END_OF_CYCLE</code></td><td>The generation of the <i>end of cycle</i> interrupts of the current SPI unit will be enabled.</td></tr> </tbody> </table>	Symbol	Meaning	<code>DIO_CLASS1_INT_END_OF_CYCLE</code>	The generation of the <i>end of cycle</i> interrupts of the current SPI unit will be enabled.		
Symbol	Meaning						
<code>DIO_CLASS1_INT_END_OF_CYCLE</code>	The generation of the <i>end of cycle</i> interrupts of the current SPI unit will be enabled.						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						

Related topics**References**

DioCl1Spi_disableInterrupts.....	456
DioCl1Spi_setInterruptMode.....	460

DioCl1Spi_setChipSelectPolarity

Syntax

```
UInt32 DioCl1Spi_setChipSelectPolarity(
    DioCl1SpiDrvObject *pDioCl1Spi,
    UInt32 ChipSelectPolarity)
```

Include file

IoDrvDioClass1Spi.h

Purpose

To set the polarity of the chip select channels of the specified SPI unit.

Description

This function specifies the signal polarity of the generated chip select signal(s). The setting applies to all the chip select channels that are controlled by the current SPI unit.

In low active mode, the connected SPI device is selected if the generated chip select signal is driven to the digital low level. In high active mode, the connected SPI device is selected if the generated chip select signal is driven to the digital high level. You can specify either the low active mode or the high active mode.

The function is intended to be called during the initialization phase of the real-time application. The setting does not take effect until **DioCl1Spi_apply** was called. If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using **DioCl1Spi_create**.

ChipSelectPolarity Lets you specify the polarity of the chip select channel(s).

Symbol	Meaning
DIO_CLASS1_SPI_CS_LOW_ACTIVE	The chip select output signal(s) are in low active mode (default).
DIO_CLASS1_SPI_CS_HIGH_ACTIVE	The chip select output signal(s) are in high active mode.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1Spi_setInterruptMode

Syntax

```
UInt32 DioCl1Spi_setInterruptMode(
    DioCl1SpiSDrvObject *pDioCl1Spi,
    UInt32 InterruptMode)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To specify the interrupt mode of the current SPI unit.

Description

This function defines whether an interrupt will be generated for the current SPI unit when an SPI cycle is finished.

In addition to interrupts, you can generate trigger signals after each SPI cycle, refer to [DioCl1Spi_setTriggerMode](#) on page 464.

The function is intended to be called during the initialization phase of the real-time application. The setting does not take effect until [DioCl1Spi_apply](#) was called. If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters	<p>pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using DioCl1Spi_create.</p> <p>InterruptMode Lets you specify whether an <i>end of cycle</i> interrupt should be generated after each SPI cycle. A related SPI cycle has to be created beforehand using DioCl1SpiCycle_create.</p>
<hr/>	
Symbol	Meaning
DIO_CLASS1_NO_INTERRUPT	No interrupt is generated (default).
DIO_CLASS1_INT_END_OF_CYCLE	An interrupt is generated after each SPI cycle.
<hr/>	
Return value	The function returns an error code.
<hr/>	
Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1Spi_SetIoIntVector

Syntax	<pre>UInt32 DioCl1Spi_SetIoIntVector(DioCl1SpiDrvObject *pDioCl1Spi, UInt32 SourceSelect, IoLibIToIntHandler IoIntHandler)</pre>
Include file	IoDrvDioClass1Spi.h
Purpose	To specify the interrupt handler processing interrupts from the SPI unit.
Description	<p>This function is used to register the function that handles the interrupts generated at the end of an SPI cycle. The interrupt generation has to be specified beforehand by using DioCl1Spi_SetInterruptMode.</p> <p>The setting takes effect not before you have called DioCl1Spi_apply during the initialization phase of the real-time application.</p> <p>If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.</p>

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using [DioCl1Spi_create](#).

SourceSelect Lets you specify the SPI interrupt source the interrupt handler is associated to.

You can only specify the *end of cycle* interrupt.

Symbol	Meaning
DIO_CLASS1_INT_END_OF_CYCLE	The <i>end of cycle</i> interrupt of the current SPI unit is associated with the interrupt handler specified in the IoIntHandler parameter.

IoIntHandler Lets you specify the pointer to the function which is to handle the generated interrupt.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1Spi_setOutputsHighZ

Syntax

```
UInt32 DioCl1Spi_setOutputsHighZ(
    DioCl1SpiDrvObject *pDioCl1Spi)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To set the state of the SPI output channels on the DIO Class 1 unit to high impedance.

Description

The digital outputs of the DIO Class 1 unit are initially set to the high impedance state (high-Z) and must be enabled for execution. With the

DioCl1Spi_setOutputsHighZ function, you can reset the outputs of the current SPI unit to the high-Z state during run time or in the termination phase of the real-time application.

The function is intended to be called during the run time of the real-time application. Nevertheless, you can also call the function during the initialization phase of the application.

The setting takes effect only after you call **DioCl1SpiCycle_start** or **DioCl1Spi_write** during the run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using **DioCl1Spi_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1Spi_setSignalVoltage

Syntax

```
UInt32 DioCl1Spi_setSignalVoltage(
    DioCl1SpiDrvObject *pDioCl1Spi,
    UInt32 SignalVoltageSelector)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To set the voltage level of the SPI output signals.

Description

This function sets the voltage level to be generated by the output channels of the specified SPI unit.

The function is intended to be called during the initialization phase of the real-time application. The setting does not take effect until **DioCl1Spi_apply** is called. If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using **DioCl1Spi_create**.

SignalVoltageSelector Lets you specify the voltage level to be generated by the output channels of the current SPI unit.

Symbol	Meaning
DIO_CLASS1_SIGNAL_2_5_V	Specifies a voltage level of 2.5 V (default).
DIO_CLASS1_SIGNAL_3_3_V	Specifies a voltage level of 3.3 V.
DIO_CLASS1_SIGNAL_5_0_V	Specifies a voltage level of 5.0 V.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1Spi_setTriggerMode

Syntax

```
UInt32 DioCl1Spi_setTriggerMode(
    DioCl1SpiDrvObject *pDioCl1Spi,
    UInt32 TriggerMode)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To specify the trigger mode of the current SPI unit.

Description

This function defines whether a trigger signal will be generated for the current SPI unit when an SPI cycle is finished.

To establish a connection between the trigger-generating component and the trigger-consuming component, you have to specify the trigger line to be used by the `DioCl1Spi_SetTriggerLineOut` function. With the `setTriggerLineIn` function of the trigger-consuming component, you enable the consuming component that is to listen to the trigger line.

In addition to trigger signals, you can generate interrupts after each SPI cycle, refer to [DioCl1Spi_SetInterruptMode](#) on page 460.

The function is intended to be called during the initialization phase of the real-time application. The setting does not take effect until `DioCl1Spi_Apply` was called. If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using `DioCl1Spi_Create`.

TriggerMode Lets you specify whether a trigger signal should be generated after each SPI cycle. A related SPI cycle has to be created beforehand using `DioCl1SpiCycle_Create`.

Symbol	Meaning
DIO_CLASS1_NO_TRIGGER	No trigger is generated (default).
DIO_CLASS1_TRIG_END_OF_CYCLE	A trigger signal is generated after each SPI cycle.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1Spi_setTriggerLineOut

Syntax

```
UInt32 DioCl1Spi_setTriggerLineOut(
    DioCl1SpiDrvObject *pDioCl1Spi,
    UInt32 TriggerLineOut)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To specify the trigger line used for trigger signals of the SPI unit.

Description

If you have specified your SPI unit to generate a trigger after each SPI cycle by using `DioCl1Spi_setTriggerMode`, you must assign a trigger line to the SPI unit by using `DioCl1Spi_setTriggerLineOut`.

A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board. You have to specify the trigger line the consumer is to listen to by using the relevant `xxx_setTriggerLineIn` function. A trigger line can be allocated only once, but multiple consumers can listen to it.

Before you can use trigger signals, you have to set the trigger mode by using `DioCl1Spi_setTriggerMode`. With `DioCl1Spi_setTriggerLineOutStatus` you enable or disable the trigger line.

The setting takes effect not before you have called `DioCl1Spi_apply` during the initialization phase of the real-time application.

If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before using `DioCl1Spi_create`.

TriggerLineOut Lets you specify the number of the trigger line to be used for transmitting the trigger signal in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.
...	...
DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	References						

DioCl1Spi_setTriggerLineOutStatus.....	467
DioCl1Spi_setTriggerMode.....	464

DioCl1Spi_setTriggerLineOutStatus

Syntax	<pre>UInt32 DioCl1Spi_setTriggerLineOutStatus(DioCl1SpiSDrvObject *pDioCl1Spi, UInt32 Status)</pre>
Include file	IoDrvDioClass1Spi.h
Purpose	To set the status of the trigger line.
Description	<p>With this function, you can enable or disable the trigger line that you specified before by using DioCl1Spi_setTriggerLineOut.</p> <p>The function is intended to be called during the initialization phase of the real-time application or during a run-to-stop or a stop-to-run transition of the real-time application.</p> <p>The setting does not take effect until you call DioCl1Spi_start.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Serial Peripheral Interface (DIO Class 1) (MicroLabBox Features).</p>

Parameters

pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using **DioCl1Spi_create**.

Status Lets you specify the status of the trigger line.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_DISABLE	Disables the trigger line.
DIO_CLASS1_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1Spi_create.....	454
DioCl1Spi_start.....	468

DioCl1Spi_start

Syntax

```
UInt32 DioCl1Spi_start(DioCl1SpiSDrvObject *pDioCl1Spi)
```

Include file

IoDrvDioClass1Spi.h

Purpose

To activate the specified SPI master based on the channels of the DIO Class 1 unit.

Description

This function starts the specified DioCl1Spi driver so that it activates its digital input and output channels. The digital output channels are enabled to generate the output signal. If you call **DioCl1Spi_setOutputsHighZ** before, the output channels stay in the high impedance state.

DioCl1Spi_start also enables the configured trigger line. If you call **DioCl1Spi_setTriggerLineOutStatus(DIO_CLASS1_TRIGGER_LINE_DISABLE)** before, the trigger line stays disabled.

The function is intended to be called during run time of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the SPI outputs that you specified by using **DioCl1Spi_create**, you must use **DioCl1Spi_start**.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using **DioCl1Spi_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1Spi_setOutputsHighZ.....	462
DioCl1Spi_setTriggerLineOutStatus.....	467

DioCl1Spi_write

Syntax

```
UInt32 DioCl1Spi_write(DioCl1SpiSDrvObject *pDioCl1Spi)
```

Include file

IoDrvDioClass1Spi.h

Purpose

To write data to the channels of the current SPI unit during run time of the real-time application.

Description	<p>During run time of the real-time application, you can specify the following values to be updated:</p> <ul style="list-style-type: none"> ▪ The state of the output channels can be set to high-Z by using DioCl1Spi_setOutputsHighZ. <p>To update the values on the channels of the current SPI unit, you must use DioCl1Spi_write.</p> <p>If an error is detected, the very first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Serial Peripheral Interface (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1Spi Lets you specify the DIO Class 1 SPI unit to be used. The DioCl1Spi driver object had to be created before by using DioCl1Spi_create.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1SpiCycle_apply

Syntax	<pre>UInt32 DioCl1SpiCycle_apply(DioCl1SpiSCycDrvObject *pDioCl1SpiCycle)</pre>
Include file	<code>IoDrvDioClass1Spi.h</code>
Purpose	To apply the initialization settings to the specified SPI cycle.
Description	<p>Before you can start the specified SPI cycle configuration, you have to release the settings of the SPI cycle driver object by using DioCl1SpiCycle_apply. You have to do this also for the default values and for values that you specified by using the DioCl1SpiCycle_set<Parameter> functions.</p> <p>To enable the transfer of SPI cycles, you must use DioCl1SpiCycle_start.</p>

The function **DioCl1SpiCycle_apply** is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

To update settings during run time, you have to use **DioCl1SpiCycle_write**.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using **DioCl1SpiCycle_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_create

Syntax

```
UInt32 DioCl1SpiCycle_create(
    DioCl1SpiSCycDrvObject **ppDioCl1SpiCycle,
    UInt32 SpiUnitNo,
    UInt32 CycleNo,
    UInt32 WordCount,
    UInt32 BitCountPerWord)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To create the I/O driver object for an SPI cycle configuration.

Description

This function performs all the steps necessary to create a specific I/O driver object for providing an SPI cycle configuration.

You can specify up to 64 SPI cycle configurations for one SPI unit. The SPI unit, which you have to specify by its number in the cycle configuration, must be created before you call this function by using `DioCl1Spi_create`.

This function sets initial values, for example, for the timing parameters, so that the driver is ready for use.

The initial values are:

- The least significant bit (bit 0) is transferred first.
- The chip select signal CS1 is activated.
- The baud rate is set to 10 kBd.
- The CLK signal (clock) is low when the transfer starts.
- The SPI master captures data at every odd-numbered (leading) edge of the CLK signal.
- The time after transfer is set to 0 s.
- The time before transfer is set to 0 s.
- The chip select inactive time is set to 1 µs.
- The time between words is set to 0 s.

To completely set up an SPI communication, you have to create the driver objects for the SPI unit and the SPI cycle with the same SPI unit number.

If the SPI driver object needed by the SPI cycle configuration is not available, the function issues an error. The function is intended to be called during the initialization phase of the real-time application. If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

Note

The maximum length of an SPI cycle is limited to 4096 bits.

You can calculate the length in bits of a cycle message using the following formula:

$$\text{CycleLength} = \text{floor}((\text{BitCountPerWord} + 31)/32) \cdot 32 \cdot \text{WordCount}$$

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

ppDioCl1SpiCycle Lets you specify the address of a variable which provides the address of the created SPI cycle driver object.

SpiUnitNo Lets you specify the number of the SPI unit to be used in the range 1 ... 4.

Symbol	Meaning
DIO_CLASS1_SPI_UNIT_1	Specifies SPI unit 1.
DIO_CLASS1_SPI_UNIT_2	Specifies SPI unit 2.
DIO_CLASS1_SPI_UNIT_3	Specifies SPI unit 3.
DIO_CLASS1_SPI_UNIT_4	Specifies SPI unit 4.

CycleNo Lets you specify the number of the cycle configuration in the range 1 ... 64. The number is used to identify the cycle when receiving or transmitting an SPI message.

WordCount Lets you specify the number of words the SPI cycle consists of in the range 1 ... 64.

BitCountPerWord Lets you specify the length in bits of one word of the SPI cycle in the range 1 ... 128.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_getCycleData

Syntax

```
UInt32 DioCl1SpiCycle_getCycleData(
    DioCl1SpiSCycDrvObject *pDioCl1SpiCycle,
    UInt32 *pCycleData)
```

Include file

IoDrvDioClass1Spi.h

Purpose

To get the SPI cycle data received by the SPI unit.

Description

The function does not get the requested cycle data directly from the SPI channels, but from an internal buffer of the driver object. To guarantee that you get the response data from the latest SPI cycle, you first have to use [DioCl1SpiCycle_read](#).

The data words of the SPI cycle are stored in an array. If a single data word spans more than one array element, there are no gaps within the elements so that

every element, except for the last one, are completely filled with relevant information. The last array element, used to hold the remaining bits of a data word according to the specified length of a data word in bits, is filled with bits of a consecutive data word and thus contains unused bits. If a data word fits into one array element, every data word of the SPI cycle is stored in a separate array element.

Note

If a single data word of the SPI cycle spans more than one array element, the portions of the data word are stored in ascending (little endian) order. This means that the least significant bits are written to the array element with the lowest index and the most significant bits are written to the array element with the highest index.

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using [DioCl1SpiCycle_create](#).

pCycleData Lets you specify the address of an array that provides the data to be received from the SPI unit according to the SPI cycle configuration.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
DRV_CL1_SPI_RX_DATA_LOST	Cycle data in the receive buffer was overwritten before being read.
DRV_CL1_SPI_RX_DATA_INCOMPLETE	The hardware FIFO buffer for the receive operation is empty.
> IOLIB_NO_ERROR	The function could not serve its purpose successfully.

DioCl1SpiCycle_read

Syntax

```
UInt32 DioCl1SpiCycle_read(
    DioCl1SpiCycleObject *pDioCl1SpiCycle)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To read cycle data from the input channels of the current SPI unit.

Description

The function stores the data of the SPI cycle configuration that has been received from the SPI unit to an internal buffer of the driver object. To read the cycle data from the internal buffer, you must use `DioCl1SpiCycle_getCycleData`.

Before you can use `DioCl1SpiCycle_read`, you have to enable the inputs and outputs of the SPI unit by using `DioCl1Spi_start` and you have to enable the transfer operation for the SPI cycle by using `DioCl1SpiCycle_start`.

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using `DioCl1SpiCycle_create`.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_setBaudRate

Syntax

```
UInt32 DioCl1SpiCycle_setBaudRate(  
    DioCl1SpiCycDrvObject *pDioCl1SpiCycle,  
    Float64 BaudRate)
```

Include file

IoDrvDioClass1Spi.h

Purpose

To set the baud rate used for transferring an SPI cycle.

Description

You can use this function to adapt the baud rate of the current SPI cycle of your SPI master to the baud rates used by the connected SPI slaves. The default baud rate of 10 kBd can be changed to a value in the range 5.0 kBd ... 2.5 MBd.

The resulting baud rate might differ from the specified value. It can be calculated by the following formula:

```
BaudRate = 25e06 / floor(25e06 / TargetBaudRate + 0.5) Bd
```

The setting takes effect not before you have called **DioCl1SpiCycle_apply** during the initialization phase of the real-time application.

If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using **DioCl1SpiCycle_create**.

BaudRate Lets you specify the baud rate to be used for transferring the cycle data in Baud in the range 5.0 kBd ... 2.5 MBd.

If the value specified is smaller than the minimum value, it is set to 5 kBd. If it is greater than the maximum value, it is set to 2.5 MBd.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_setBitDirection

Syntax

```
UInt32 DioCl1SpiCycle_setBitDirection(
    DioCl1SpiSCycDrvObject *pDioCl1SpiCycle,
    UInt32 BitDirectionSelector)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To set the bit order of an SPI cycle when transferring data.

Description

The bit direction of a cycle configuration specifies whether the least or the most significant bit of a word is output first at the MOSI signal of the SPI master.

The setting takes effect not before you have called **DioCl1SpiCycle_apply** during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using **DioCl1SpiCycle_create**.

BitDirectionSelector Lets you specify the bit order applied for transferring the cycle data.

Symbol	Meaning
DIO_CLASS1_SPI_DIR_LSB_TO_MSB	The least significant bit of a word is transferred first (default).
DIO_CLASS1_SPI_DIR_MSB_TO_LSB	The most significant bit of a word is transferred first.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_setChipSelectConf

Syntax

```
UInt32 DioCl1SpiCycle_setChipSelectConf(
    DioCl1SpiSCycDrvObject *pDioCl1SpiCycle,
    UInt32 ChipSelectConfiguration)
```

Include file

IoDrvDioClass1Spi.h

Purpose

To set the chip select configuration of an SPI cycle.

Description

The chip select configuration controls which chip select outputs of the SPI unit are activated when the relevant SPI cycle is transferred. Its value is a bit mask in the range 1 ... 15. For example, if you have specified four chip select channels, you can address up to 15 channels using a decoder. For example, by specifying `ChipSelectConfiguration = 0x000B`, you activate the chip select channels 1, 2, and 4. If you do not use a decoder, you can address up to four channels with the four bits of the chip select configuration.

The setting takes effect not before you have called `DioCl1SpiCycle_apply` during the initialization phase of the real-time application.

If an error is detected, the very first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using [DioCl1SpiCycle_create](#).

ChipSelectConfiguration Lets you specify the bit mask to be applied to the chip select outputs. To specify more than one chip select output signal to be activated, combine the predefined symbols by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_SPI_MASK_CS1	The CS1 output signal is to be activated when transferring the SPI cycle.
DIO_CLASS1_SPI_MASK_CS2	The CS2 output signal is to be activated when transferring the SPI cycle.
DIO_CLASS1_SPI_MASK_CS3	The CS3 output signal is to be activated when transferring the SPI cycle.
DIO_CLASS1_SPI_MASK_CS4	The CS4 output signal is to be activated when transferring the SPI cycle.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_setClockPolarity

Syntax

```
UInt32 DioCl1SpiCycle_setClockPolarity(
    DioCl1SpiSocyDrvObject *pDioCl1SpiCycle,
    UInt32 ClockPolarity)
```

Include file

IoDrvDioClass1Spi.h

Purpose

To set the polarity of the clock signal.

Description

You can use this function to change the default value of the clock polarity that is set when you create the SPI cycle using [DioCl1SpiCycle_create](#).

The clock polarity (CPOL) specifies the initial signal level of the clock (CLK) output signal when a transfer of the specified SPI cycle is started.

For further information on the clock parameters, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

The setting takes effect not before you have called `DioCl1SpiCycle_apply` during the initialization phase of the real-time application.

If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using `DioCl1SpiCycle_create`.

ClockPolarity Lets you specify the polarity of the SPI clock at the start of a cycle.

Symbol	Meaning
DIO_CLASS1_SPI_CPOL_0	The CLK signal is low when transfer starts (default).
DIO_CLASS1_SPI_CPOL_1	The CLK signal is high when transfer starts.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_setClockPhase

Syntax

```
UInt32 DioCl1SpiCycle_setClockPhase(
    DioCl1SpiSCycDrvObject *pDioCl1SpiCycle,
    UInt32 ClockPhase)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose	To set the phase of the clock signal.	
Description	<p>You can use this function to change the default value of the clock phase that is set when you create the SPI cycle using DioCl1SpiCycle_create.</p> <p>The clock phase (CPHA) specifies the points in time at which the clock signal of the SPI unit has to output and capture the bits of the specified SPI cycle.</p> <p>For further information on the clock parameters, refer to Serial Peripheral Interface (DIO Class 1) (MicroLabBox Features).</p> <p>The setting takes effect not before you have called DioCl1SpiCycle_apply during the initialization phase of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.</p>	
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Serial Peripheral Interface (DIO Class 1) (MicroLabBox Features).</p>	
Parameters	<p>pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using DioCl1SpiCycle_create.</p> <p>ClockPhase Lets you specify the phase of the SPI clock to start the data capture.</p>	
Symbol	Meaning	
DIO_CLASS1_SPI_CPHA_0	Output data on trailing (even-numbered) edges of the clock signal. Capture data on leading (odd-numbered) edges of the clock signal (default).	
DIO_CLASS1_SPI_CPHA_1	Output data on leading (odd-numbered) edges of the clock signal. Capture data on trailing (even-numbered) edges of the clock signal.	
Clock Polarity	Whether the leading or trailing edges are rising or falling edges depends on the ClockPolarity parameter that is set by using DioCl1SpiCycle_setClockPolarity .	
DIO_CLASS1_SPI_CPOL_0	Rising edge	Falling edge
DIO_CLASS1_SPI_CPOL_1	Falling edge	Rising edge

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_setCycleData

Syntax

```
UInt32 DioCl1SpiCycle_setCycleData(
    DioCl1SpiSCycDrvObject *pDioCl1SpiCycle,
    UInt32 *pCycleData)
```

Include file

IoDrvDioClass1Spi.h

Purpose

To set the SPI cycle data to be transferred.

Description

The data words of the SPI cycle must be stored in an array. If a single data word spans more than one array element, there must not be any gaps within the elements. This means that every element, except for the last one, must be completely filled with relevant information. The last array element, used to hold the remaining bits of a data word according to the specified length of a data word in bits, must not be filled with bits of a consecutive data word and thus can contain unused bits. If a data word fits into one array element, every data word of the SPI cycle must be stored in a separate array element.

Note

If a single data word of the SPI cycle spans more than one array element, the portions of the data word must be stored in ascending (little endian) order. This means that the least significant bits must be written to the array element with the lowest index and the most significant bits must be written to the array element with the highest index.

The cycle data is not transferred by the SPI unit until you have called [DioCl1SpiCycle_write](#).

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using [DioCl1SpiCycle_create](#).

pCycleData Lets you specify the address of an array that provides the data to be transferred by the SPI unit according to the SPI cycle configuration.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
DRV_CL1_SPI_TX_DATA_LOST	Cycle data in the transmit buffer were overwritten before being sent.
DRV_CL1_SPI_TX_FIFO_OVERFLOW	The hardware FIFO buffer for the transmit operation is full.
> IOLIB_NO_ERROR	The function could not serve its purpose successfully.

DioCl1SpiCycle_setTimeAfterTran

Syntax

```
UInt32 DioCl1SpiCycle_setTimeAfterTran(
    DioCl1SpiCycDrvObject *pDioCl1SpiCycle,
    Float64 TimeAfterTransfer)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To set the time after transfer value of an SPI cycle.

Description

You can use this function to change the default value of the timing parameter *time after transfer* (TAT) that is set when you create the SPI cycle using [DioCl1SpiCycle_create](#).

The TAT timing parameter defines the time interval from the end of the last period of the clock signal of the SPI master to the point in time at which the chip select outputs are driven to the inactive signal state. Whether the TAT value is applied once per SPI cycle or once per word depends on the *time between words* parameter specified by using [DioCl1SpiCycle_setTimeBetwWords](#). The value range is internally divided into 12 subranges with different step widths. If you specify a value that is between two consecutive values, it is saturated to the next larger value.

For further information on the timing parameters, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

The setting takes effect not before you have called `DioCl1SpiCycle_apply` during the initialization phase of the real-time application.

If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using `DioCl1SpiCycle_create`.

TimeAfterTransfer Lets you specify the time after transfer in seconds in the range 0 ... 800 µs.

This is the time between the end of the last period and the deactivation of the chip select signal (TimeBetweenWords = 0) or the cycle (TimeBetweenWords > 0) of the clock signal is ended.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_setTimeBeforeTran

Syntax

```
UInt32 DioCl1SpiCycle_setTimeBeforeTran(
    DioCl1SpiSCycDrvObject *pDioCl1SpiCycle,
    Float64 TimeBeforeTransfer)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To set the time before transfer value of an SPI cycle.

Description	You can use this function to change the default value of the timing parameter <i>time before transfer</i> (TBT) that is set when you create the SPI cycle using DioCl1SpiCycle_create .
--------------------	--

The TBT timing parameter defines the time interval from the point in time at which the chip select outputs are driven to the active signal state to the point in time at which the first period of the clock signal of the SPI unit is started.

Whether the TBT value is applied once per SPI cycle or once per word depends on the *time between words* parameter specified by using

DioCl1SpiCycle_setTimeBetweenWords. The value range is internally divided into 12 subranges with different step widths. If you specify a value that is between two consecutive values, it is saturated to the next larger value.

For further information on the timing parameters, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

The setting takes effect not before you have called **DioCl1SpiCycle_apply** during the initialization phase of the real-time application.

If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) .
	For more information on the I/O mapping, refer to Serial Peripheral Interface (DIO Class 1) (MicroLabBox Features) .

Parameters	pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using DioCl1SpiCycle_create . TimeBeforeTransfer Lets you specify the time before transfer in seconds in the range 0 ... 800 µs. This is the time between the activation of the chip select signal and the beginning of the first period (TimeBetweenWords = 0) or the cycle (TimeBetweenWords > 0) of the clock signal is started.
-------------------	--

Return value	The function returns an error code.
<hr/>	
Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_setTimeBetwWords

Syntax

```
UInt32 DioCl1SpiCycle_setTimeBetwWords(
    DioCl1SpiCycDrvObject *pDioCl1SpiCycle,
    Float64 TimeBetweenWords)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To set the time between words value for an SPI cycle.

Description

You can use this function to change the default value of the *time between words* (TBW) timing parameter that is set when you create the SPI cycle using `DioCl1SpiCycle_create`.

The TBW timing parameter defines the optional waiting time between two consecutive words of an SPI cycle during transfer. The TBW value affects the entire timing behavior of an SPI cycle configured by the `TimeAfterTransfer`, `TimeBeforeTransfer`, and `TimeChipSelectInactive` parameters.

The value range is internally divided into 12 subranges with different step widths. If you specify a value that is between two consecutive values, it is saturated to the next larger value.

For further information on the timing parameters, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

The setting takes effect not before you have called `DioCl1SpiCycle_apply` during the initialization phase of the real-time application.

If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using `DioCl1SpiCycle_create`.

TimeBetweenWords Lets you specify the optional waiting time between two consecutive words of an SPI cycle in seconds in the range 20 ns ... 800 µs or 0 µs.

If the value of this parameter is greater than zero, this value specifies the time between the last bit of the current word of the MOSI signal and the first bit of the following word of the MOSI signal. The chip select signal will be paused for the specified time instead of being negated between two consecutive words of an SPI cycle.

If the value of this parameter is zero, the timing behavior of the transfer is specified by the `TimeAfterTransfer`, `TimeBeforeTransfer`, and `CSInactiveTime` parameters. The chip select signal will switch to the inactive state between the word transmissions.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

DioCl1SpiCycle_setTimeCsInactive

Syntax

```
UInt32 DioCl1SpiCycle_setTimeCsInactive(
    DioCl1SpiSCycDrvObject *pDioCl1SpiCycle,
    Float64 TimeChipSelectInactive)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To set the minimum inactive time of the chip select signal.

Description

You can use this function to change the default value of the *time chip select inactive* (TCS) timing parameter that is set when you create the SPI cycle using `DioCl1SpiCycle_create`.

The TCS timing parameter defines the minimum time interval during which the chip select outputs are driven to the inactive signal state between two consecutive SPI cycles. If applicable, it also defines the time interval during which the chip select outputs are driven to the inactive signal state between two consecutive words within an SPI cycle. Whether the TCS value is applied between two SPI cycles or between two words depends on the *time between words* parameter specified by using `DioCl1SpiCycle_setTimeBetwWords`. The value range is internally divided into 12 subranges with different step widths. If you specify a value that is between two consecutive values, it is saturated to the next larger value.

For further information on the timing parameters, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

The setting takes effect not before you have called `DioCl1SpiCycle_apply` during the initialization phase of the real-time application.

If an error is detected, the first instance of the error is returned to the caller and a relevant error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

`pDioCl1SpiCycle` Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using `DioCl1SpiCycle_create`.

`TimeChipSelectInactive` Lets you specify the minimum inactive time of the chip select signal between two cycles in seconds in the range 20 ns ... 800 µs. This parameter is also used to specify the time interval during which the chip select signal is to be driven to inactive state between two words of a chip select cycle.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

DioCl1SpiCycle_start

Syntax

```
UInt32 DioCl1SpiCycle_start(
    DioCl1SpiSCycDrvObject *pDioCl1SpiCycle)
```

Include file

`IoDrvDioClass1Spi.h`

Purpose

To enable the transfer of SPI cycle data.

Description	The transfer of SPI cycle data must be enabled before you can write or read cycle data from the digital input and output channels of the specified SPI unit. The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.						
I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) . For more information on the I/O mapping, refer to Serial Peripheral Interface (DIO Class 1) (MicroLabBox Features) .						
Parameters	pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using DioCl1SpiCycle_create .						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

DioCl1SpiCycle_write

Syntax	<pre>UInt32 DioCl1SpiCycle_write(DioCl1SpiSCycDrvObject *pDioCl1SpiCycle)</pre>
Include file	<code>IoDrvDioClass1Spi.h</code>
Purpose	To transfer the cycle data to the output channels of the current SPI unit.
Description	The data that you specified for an SPI cycle by using DioCl1SpiCycle_setCycleData are output by the SPI unit according to the SPI cycle configuration. Before you can use this function, you have to enable the outputs of the SPI unit by using DioCl1Spi_start and you have to enable the transfer operation for the SPI cycle by using DioCl1SpiCycle_start .

The function is intended to be called during run time of the real-time application. If an error is detected, the very first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Serial Peripheral Interface \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1SpiCycle Lets you specify the SPI cycle configuration to be used. The DioCl1SpiCycle driver object had to be created before by using [DioCl1SpiCycle_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Electric Motor Control Functions

Introduction

You are informed about the advanced functions provided by the MicroLabBox for controlling electric motors.

Where to go from here

Information in this section

Functions for the angle computation unit managing the sensor inputs and actuator outputs.

[Angle Computation Unit \(ACU\)](#)..... 493

Functions for the sensor interfaces.

[EnDat Interface](#)..... 536

[Hall Sensor Interface](#)..... 568

[Incremental Encoder Interface](#)..... 604

[Resolver Interface](#)..... 667

[SSI Interface](#)..... 686

Functions for the actuator interfaces.

[Block-Commutated PWM Generation](#)..... 725

[Multichannel PWM Signal Generation](#)..... 771

Information in other sections

[RTI Electric Motor Control Blockset Reference](#)

Provides concise information on the blocks of the RTI Electric Motor Control Blockset.

[Electric Motor Control \(MicroLabBox Features !\[\]\(9110080ffa3616349c2c57e931ef80e7_img.jpg\)](#)

MicroLabBox provides specific I/O features used for electric motor control.

Angle Computation Unit (ACU)

Introduction

You can use an angle computation unit to generate events at absolute or relative sensor positions. By attaching two position sensors of different types to an ACU, you can compensate their type-specific disadvantages.

For more information on ACUs, refer to [Electric Motor Control \(MicroLabBox Features\)](#).

Where to go from here

Information in this section

Acu_apply	495
To apply the initialization settings of an angle computation unit.	
Acu_create	496
To create the I/O driver object for an angle computation unit.	
Acu_disableInterrupts	498
To disable the interrupt generation of an ACU.	
Acu_enableInterrupts	499
To enable the interrupt generation of an ACU.	
Acu_getCurrentInputSensorNo	501
To get the sensor that is currently used by the ACU.	
Acu_getElecPosition	502
To get the current electrical position calculated by the ACU.	
Acu_getIsElecPosValid	503
To get the validity state of the electrical position calculated by the ACU.	
Acu_getIsMechPosValid	504
To get the validity state of the mechanical position calculated by the ACU.	
Acu_getIsMotorSpeedValid	505
To get the validity state of the motor speed calculated by the ACU.	
Acu_getIsSensorElecPosValid	506
To get the validity state of the electrical position provided by a sensor to an ACU.	
Acu_getMechPosition	508
To get the current mechanical position calculated by the ACU.	
Acu_getMotorSpeed	509
To get the current motor speed calculated by the ACU.	
Acu_getSensorElecPosition	510
To get the electrical position provided by a sensor to an ACU.	

Acu_read	511
To read data from an angle computation unit.	
Acu_setAbsolutePositionForEvent	513
To specify an absolute angle position for the event generation of an ACU.	
Acu_setInputSensor	514
To attach a position-generating sensor to an ACU.	
Acu_setInterruptMode	516
To specify the interrupt mode for an ACU.	
Acu_setIntVector	518
To set the interrupt handler for interrupts from an ACU.	
Acu_setMinMotorSpeed	519
To specify the minimal motor speed.	
Acu_setMotorPolePairCount	520
To specify the motor's number of pole pairs.	
Acu_setPositionEventBase	521
To select the position type to be used for event generation of an ACU.	
Acu_setPositionEventHysteresis	523
To set the hysteresis of the position that is relevant for the event generation of an ACU.	
Acu_setRelativePositionForEvent	524
To specify the relative angle for periodic event generation of an ACU.	
Acu_setRelativePositionOffset	525
To specify an offset angle for periodic event generation of an ACU.	
Acu_setSensorForPositionEvent	526
To select the position-generating sensor as the base for event generation of an ACU.	
Acu_setSensorPolePairCount	527
To specify the sensor's number of pole pairs.	
Acu_setTriggerLineOut	529
To select the trigger line used for trigger event generation of an ACU.	
Acu_setTriggerLineOutStatus	530
To set the status of the trigger line.	
Acu_setTriggerMode	531
To specify the trigger mode for an ACU.	
Acu_start	533
To start an angle computation unit.	
Acu_write	534
To update the settings of an angle computation unit during run time.	

Acu_apply

Syntax

```
UInt32 Acu_apply(AcuSDrvObject *pAcu)
```

Include file

IoDrvAcu.h

Purpose

To apply the initialization settings of an angle computation unit.

Description

Before you can start the specified ACU, you must transfer the settings of the ACU driver object to the hardware by using **Acu_apply**.

You must do this also for the default values and for values that you specified via the **Acu_set<Parameter>** functions. The **Acu_apply** function is intended to be called at the end of the initialization phase of the real-time application.

The angle computation unit is not activated until you have called **Acu_start**.

To update settings during run time, you must use **Acu_write**.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_create.....	496
Acu_SetInputSensor.....	514
Acu_SetInterruptMode.....	516
Acu_SetToIntVector.....	518
Acu_SetMotorPolePairCount.....	520
Acu_SetPositionEventBase.....	521
Acu_SetRelativePositionForEvent.....	524
Acu_SetRelativePositionOffset.....	525
Acu_SetSensorForPositionEvent.....	526
Acu_SetSensorPolePairCount.....	527

Acu_setTriggerLineOut.....	529
Acu_setTriggerMode.....	531
Acu_start.....	533
Acu_write.....	534

Acu_create

Syntax

```
UInt32 Acu_create(
    AcuSDrvObject **ppAcu,
    UInt32 AngleComputationUnitNo)
```

Include file

IoDrvAcu.h

Purpose

To create the I/O driver object for an angle computation unit.

Description

This function is used to perform all the steps necessary to create an I/O driver object for an angle computation unit (ACU).

An ACU processes data that is captured by up to two attached position-generating sensors that you have created and configured. The captured position data is used for the following purposes:

- Events (trigger signals or interrupts) can be generated at specified absolute or relative sensor positions.
- You can configure the output function of a block-commutated motor control unit to receive angle values immediately to relieve the simulator's calculating node.
- You can combine two attached sensors to compensate the specific disadvantages of their sensor types.

For example, Hall sensors can immediately provide an absolute position but have a lower resolution. Incremental encoders have a higher resolution but need an index pulse to provide an absolute position. Via an ACU you can combine both sensors, so that the Hall sensor provides the position value before the occurrence of the index pulse and the incremental encoder afterwards.

If you use an absolute encoder, you can attach it to Sensor B as a single sensor.

- For Hall sensors the ACU provides extrapolated electrical positions in between the values captured by the attached sensors.

For more information on the use of ACUs, refer to [Electric Motor Control \(MicroLabBox Features\)](#).

The ACU driver is initialized with default values. It is ready for use after you attached at least one of the sensors *Sensor A* or *Sensor B* via [**Acu_setInputSensor**](#).

The initial values are:

- Sensor A: No sensor attached
- Number of pole pairs of sensor A: 1
- Sensor B: No sensor attached
- Number of pole pairs of sensor B: 1
- Extrapolation: Activated
- Number of pole pairs of the motor: 1
- Basis for events on positions: Electrical position of the motor
- Sensor used for events on positions: None
- Event on absolute angle position 1: Disabled
- Event on absolute angle position 2: Disabled
- Event on absolute angle position 3: Disabled
- Event on absolute angle position 4: Disabled
- Event on relative angle position: Disabled
- Position offset for event generation on relative position: 0.0°

The function is intended to be called during the initialization phase of the real-time application.

You can use the [**Acu_set<Parameter>**](#) functions to change the default parameters. Before you can start the specified ACU, you have to transfer the settings of the ACU driver object to the hardware by using [**Acu_apply**](#).

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters

ppAcu Lets you specify the address of a variable which holds the address of the created driver object for the angle computation unit.

AngleComputationUnitNo Lets you specify the instance number of the ACU in the range 1 ... 4. One driver object can handle one angle computation unit. You can use the instance number to attach the ACU to a block-commutated motor control unit via [**DioCl1BcPwmOut_setAcu**](#).

Symbol	Meaning
ACU_INSTANCE_1	Specifies the angle computation unit number 1.
...	...
ACU_INSTANCE_4	Specifies the angle computation unit number 4.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References																										
	<table border="0"> <tr> <td>Acu_apply.....</td> <td>495</td> </tr> <tr> <td>Acu_setInputSensor.....</td> <td>514</td> </tr> <tr> <td>Acu_setInterruptMode.....</td> <td>516</td> </tr> <tr> <td>Acu_setIoIntVector.....</td> <td>518</td> </tr> <tr> <td>Acu_setMotorPolePairCount.....</td> <td>520</td> </tr> <tr> <td>Acu_setPositionEventBase.....</td> <td>521</td> </tr> <tr> <td>Acu_setRelativePositionForEvent.....</td> <td>524</td> </tr> <tr> <td>Acu_setRelativePositionOffset.....</td> <td>525</td> </tr> <tr> <td>Acu_setSensorForPositionEvent.....</td> <td>526</td> </tr> <tr> <td>Acu_setSensorPolePairCount.....</td> <td>527</td> </tr> <tr> <td>Acu_setTriggerLineOut.....</td> <td>529</td> </tr> <tr> <td>Acu_setTriggerMode.....</td> <td>531</td> </tr> <tr> <td>Acu_start.....</td> <td>533</td> </tr> </table>	Acu_apply.....	495	Acu_setInputSensor.....	514	Acu_setInterruptMode.....	516	Acu_setIoIntVector.....	518	Acu_setMotorPolePairCount.....	520	Acu_setPositionEventBase.....	521	Acu_setRelativePositionForEvent.....	524	Acu_setRelativePositionOffset.....	525	Acu_setSensorForPositionEvent.....	526	Acu_setSensorPolePairCount.....	527	Acu_setTriggerLineOut.....	529	Acu_setTriggerMode.....	531	Acu_start.....	533
Acu_apply.....	495																										
Acu_setInputSensor.....	514																										
Acu_setInterruptMode.....	516																										
Acu_setIoIntVector.....	518																										
Acu_setMotorPolePairCount.....	520																										
Acu_setPositionEventBase.....	521																										
Acu_setRelativePositionForEvent.....	524																										
Acu_setRelativePositionOffset.....	525																										
Acu_setSensorForPositionEvent.....	526																										
Acu_setSensorPolePairCount.....	527																										
Acu_setTriggerLineOut.....	529																										
Acu_setTriggerMode.....	531																										
Acu_start.....	533																										

Acu_disableInterrupts

Syntax	<pre>UInt32 Acu_disableInterrupts(AcuSDrvObject *pAcu, UInt32 InterruptSelect)</pre>
Include file	IoDrvAcu.h
Purpose	To disable the interrupt generation of an ACU.
Description	<p>With this function, you can disable the generation of interrupts at absolute positions that you specified via Acu_setAbsolutePositionForEvent or at relative positions that you specified via Acu_setRelativePositionForEvent.</p> <p>You must have specified the interrupt mode before by using Acu_setInterruptMode.</p> <p>The function is intended to be called during the run time of the real-time application.</p>

Settings are only effective if the driver object functionality was activated before by using **Acu_start**.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

InterruptSelect Lets you select the interrupt to be disabled. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
ACU_INT_ABSOLUTE_POSITION_1	Disables the generation of interrupts at absolute position 1.
...	...
ACU_INT_ABSOLUTE_POSITION_4	Disables the generation of interrupts at absolute position 4.
ACU_INT_RELATIVE_POSITION	Disables the generation of interrupts at each relative position.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_apply.....	495
Acu_create.....	496

Acu_enableInterrupts

Syntax

```
UInt32 Acu_enableInterrupts(
    AcuSDrvObject *pAcu,
    UInt32 InterruptSelect)
```

Include file

IoDrvAcu.h

Purpose

To enable the interrupt generation of an ACU.

Description	<p>With this function, you can either enable the generation of interrupts or enable the generation of interrupts at absolute positions that you specified via Acu_setAbsolutePositionForEvent or at relative positions that you specified via Acu_setRelativePositionForEvent.</p> <p>You must have specified the interrupt mode before by using Acu_setInterruptMode.</p> <p>After calling Acu_start, interrupts are disabled.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>Settings are only effective if the driver object functionality was activated before by using Acu_start.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>
--------------------	---

Parameters	<p>pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using Acu_create.</p> <p>InterruptSelect Lets you select the interrupt to be enabled. The symbols can be combined by using the bitwise OR operator.</p>
-------------------	--

Symbol	Meaning
ACU_INT_ABSOLUTE_POSITION_1	Enables the generation of interrupts at absolute position 1.
...	...
ACU_INT_ABSOLUTE_POSITION_4	Enables the generation of interrupts at absolute position 4.
ACU_INT_RELATIVE_POSITION	Enables the generation of interrupts at each relative position.

Return value	The function returns an error code.
<hr/>	
Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References								
<table border="0"> <tr> <td>Acu_apply.....</td> <td>495</td> </tr> <tr> <td>Acu_create.....</td> <td>496</td> </tr> <tr> <td>Acu_setAbsolutePositionForEvent.....</td> <td>513</td> </tr> <tr> <td>Acu_setRelativePositionForEvent.....</td> <td>524</td> </tr> </table>		Acu_apply.....	495	Acu_create.....	496	Acu_setAbsolutePositionForEvent.....	513	Acu_setRelativePositionForEvent.....	524
Acu_apply.....	495								
Acu_create.....	496								
Acu_setAbsolutePositionForEvent.....	513								
Acu_setRelativePositionForEvent.....	524								

Acu_getCurrentInputSensorNo

Syntax

```
UInt32 AcuGetCurrentInputSensorNo(
    AcuSDrvObject *pAcu,
    UInt32 *pCurrentInputSensorSelection)
```

Include file

`IoDrvAcu.h`

Purpose

To get the sensor that is currently used by the ACU.

Description

Up to two position-generating sensors (*Sensor A* and *Sensor B*) can be attached to an ACU. The ACU chooses a sensor based on the validity of the provided positions.

If two sensors are attached and one of them provides position values marked as invalid, the ACU automatically uses the position values of the other sensor.

If both sensors provide valid position values, *Sensor B* is used. Therefore, it is recommended to attach the sensor with the higher resolution as *Sensor B*.

The values are taken from the internal buffer of the ACU's driver object. However, newer values might be available. Therefore, call **Acu_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

pCurrentInputSensorSelection Lets you specify the address of a variable that states which position-generating sensor is currently used by the ACU.

Symbol	Meaning
ACU_SENSOR_A	Specifies that <i>Sensor A</i> is used.
ACU_SENSOR_B	Specifies that <i>Sensor B</i> is used.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply	495
Acu_create	496

Acu_getElecPosition

Syntax

```
UInt32 Acu_getElecPosition(
    AcuSDrvObject *pAcu,
    Float64 *pElecPosition)
```

Include file

IoDrvAcu.h

Purpose

To get the current electrical position calculated by the ACU.

Description

The angle computation unit calculates the electrical position from values that are provided by up to two attached position-generating sensors (*Sensor A* and *Sensor B*).

If the electrical position cannot be calculated, for example, if both sensors provide invalid positions, the validity state of the electrical position is set to invalid. You can check the validity state via [Acu_getIsElecPosValid](#).

For more information on the validity of position values that are returned from an ACU, refer to [Electric Motor Control \(MicroLabBox Features\)](#).

For Hall sensors only, the ACU extrapolates the electrical position.

You can identify which sensor is currently used via [Acu_getCurrentInputSensorNo](#).

The values are taken from the internal buffer of the ACU's driver object. However, newer values might be available. Therefore, call [Acu_read](#) to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using [Acu_create](#).

pElecPosition Lets you specify the address of a variable that holds the current electrical rotor position provided by the specified ACU in degrees in the range 0° ... 359.999999°. The resolution is 0.00000134°.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply.....	495
Acu_create.....	496
Acu_getIsElecPosValid.....	503

Acu_getIsElecPosValid

Syntax

```
UInt32 Acu_getIsElecPosValid(
    AcuSDrvObject *pAcu,
    UInt32 *pIsElecPositionValid)
```

Include file

IoDrvAcu.h

Purpose

To get the validity state of the electrical position calculated by the ACU.

Description

The electrical position calculated by the ACU is valid if the currently used sensor provides a position value that is marked as valid. For an example, refer to [DioCl1HallIn_getIsPositionValid](#) on page 573.

The values are taken from the internal buffer of the ACU's driver object. However, newer values might be available. Therefore, call **Acu_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

plsElecPositionValid Lets you specify the address of a variable that states whether the electrical position provided by the specified ACU is marked as valid.

Value	Meaning
0	The electrical rotor position value is invalid.
> 0	The electrical rotor position value is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply	495
Acu_create	496

Acu_getIsMechPosValid

Syntax

```
UInt32 Acu_getIsMechPosValid(
    AcuSDrvObject *pAcu,
    UInt32 *pIsMechPositionValid)
```

Include file

IoDrvAcu.h

Purpose

To get the validity state of the mechanical position calculated by the ACU.

Description

The calculated mechanical position is valid in the following cases:

- A sensor that provides a valid mechanical position is attached to the ACU. This could be an incremental encoder after its index signal occurred.
- A sensor that provides a valid electrical position, such as a Hall sensor, is attached to the ACU and the number of pole pairs of the sensor equals one.

The values are taken from the internal buffer of the ACU's driver object. However, newer values might be available. Therefore, call **Acu_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using [Acu_create](#).

pIsMechPositionValid Lets you specify the address of a variable that states whether the mechanical position provided by the specified ACU is marked as valid.

Value	Meaning
0	The mechanical rotor position value is invalid.
> 0	The mechanical rotor position value is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply	495
Acu_create	496

Acu_getIsMotorSpeedValid

Syntax

```
UInt32 Acu_getIsMotorSpeedValid(
    AcuSDrvObject *pAcu,
    UInt32 *pIsMotorSpeedValid)
```

Include file

IoDrvAcu.h

Purpose

To get the validity state of the motor speed calculated by the ACU.

Description

The motor speed calculated by the ACU is valid if the currently used sensor provides a position value that is marked as valid. For an example, refer to [DioCl1HallIn_getIsPositionValid](#) on page 573.

The values are taken from the internal buffer of the ACU's driver object. However, newer values might be available. Therefore, call **Acu_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

plsMotorSpeedValid Lets you specify the address of a variable that states whether the motor speed provided by the specified ACU is marked as valid.

Value	Meaning
0	The motor speed value is invalid.
> 0	The motor speed value is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply.....	495
Acu_create.....	496

Acu_getIsSensorElecPosValid

Syntax

```
UInt32 Acu_getIsSensorElecPosValid(
    AcuSDrvObject *pAcu,
    UInt32 AcuSensorSelector,
    UInt32 *pIsElecPositionValid)
```

Include file

IoDrvAcu.h

Purpose

To get the validity state of the electrical position provided by a sensor to an ACU.

Description

You can select one of the sensors that are attached to the ACU to specify which sensor to get the information from. The sensor itself provides the information on whether the signal is valid or not.

The values are taken from the internal buffer of the ACU's driver object. However, newer values might be available. Therefore, call **Acu_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

AcuSensorSelector Lets you select the position-generating sensor.

Symbol	Meaning
ACU_SENSOR_NONE	No sensor is selected.
ACU_SENSOR_A	To get the information from <i>Sensor A</i> .
ACU_SENSOR_B	To get the information from <i>Sensor B</i> .

plsElecPositionValid Lets you specify the address of a variable that states whether the electrical position provided by the selected sensor is marked as valid.

Value	Meaning
0	The electrical position of the sensor is invalid.
> 0	The electrical position of the sensor is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply	495
Acu_create	496

Acu_getMechPosition

Syntax

```
UInt32 Acu_getMechPosition(
    AcuSDrvObject *pAcu,
    Float64 *pMechPosition)
```

Include file

`IoDrvAcu.h`

Purpose

To get the current mechanical position calculated by the ACU.

Description

This function provides the calculated mechanical position only if at least one of the following conditions is fulfilled:

- A sensor that provides the mechanical position is attached to the ACU. This could be an incremental encoder after its index signal occurred.
- A sensor that provides the electrical position, such as a Hall sensor, is attached to the ACU and the number of pole pairs of the sensor equals one.

If the mechanical position cannot be calculated, the validity state of the mechanical position is set to invalid. You can check the validity state via [Acu_getIsMechPosValid](#).

The values are taken from the internal buffer of the ACU's driver object. However, newer values might be available. Therefore, call [Acu_read](#) to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using [Acu_create](#).

pMechPosition Lets you specify the address of a variable that holds the current mechanical rotor position that is provided by the specified ACU in degrees in the range 0° ... 359.999999°. The resolution is 0.00000134°.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply.....	495
Acu_create.....	496
Acu_getIsMechPosValid.....	504
Acu_setSensorPolePairCount.....	527

Acu_getMotorSpeed

Syntax

```
UInt32 Acu_getMotorSpeed(
    AcuSDrvObject *pAcu,
    Float64 *pMotorSpeed)
```

Include file

IoDrvAcu.h

Purpose

To get the current motor speed calculated by the ACU.

Description

The ACU calculates the rotational speed of the motor in revolutions per minute based on the electrical position. The electrical position is provided by the sensor that is selected by the ACU.

You can check whether the provided motor speed is valid via **Acu_getIsMotorSpeedValid**.

The values are taken from the internal buffer of the ACU's driver object. However, newer values might be available. Therefore, call **Acu_read** to update the internal buffer with the latest values.

Note

The speed value is calculated from the current position. Because the calculation takes some nanoseconds, the speed value might not be updated in the same model step as the position value.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

pMotorSpeed Lets you specify the address of a variable that holds the rotational speed in revolutions per minute in the range -262,143.984375 rpm ... +262,143.984375 rpm. The resolution is 0.015625 rpm.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply.....	495
Acu_create.....	496
Acu_getIsMotorSpeedValid.....	505

Acu_getSensorElecPosition

Syntax

```
UInt32 Acu_getSensorElecPosition(
    AcuSDrvObject *pAcu,
    UInt32 AcuSensorSelector,
    Float64 *pElecPosition)
```

Include file

IoDrvAcu.h

Purpose

To get the electrical position provided by a sensor to an ACU.

Description

You can select one of the sensors that are attached to the ACU to specify which sensor to get the information from.

The values are taken from the internal buffer of the ACU's driver object. However, newer values might be available. Therefore, call **Acu_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using [Acu_create](#).

AcuSensorSelector Lets you select the position-generating sensor.

Symbol	Meaning
ACU_SENSOR_NONE	No sensor is selected.
ACU_SENSOR_A	To get the electrical position from Sensor A.
ACU_SENSOR_B	To get the electrical position from Sensor B.

pElecPosition Lets you specify the address of a variable that holds the current electrical position that is provided by the specified sensor in degrees in the range $0^\circ \dots 359.999999^\circ$. The resolution is 0.00000134° .

The measured electrical position is the electrical angle value related to the motor, i.e., the measurement depends on the motor's number of pole pairs.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply	495
Acu_create	496

Acu_read

Syntax

```
UInt32 Acu_read(AcuSDrvObject *pAcu)
```

Include file

IoDrvAcu.h

Purpose

To read data from an angle computation unit.

Description

This function is used to read the result data from the specified ACU hardware and stores it as a consistent data set in an internal buffer of the driver.

The angle computation unit must be started before via **Acu_start**.

To get the values from the internal buffer, you can use one of the following functions:

- **Acu_getCurrentInputSensorNo** to get the sensor that is currently used by the ACU.
- **Acu_getElecPosition** to get the current electrical position calculated by the ACU.
- **Acu_getIsElecPosValid** to get the validity state of the electrical position calculated by the ACU.
- **Acu_getIsMechPosValid** to get the validity state of the mechanical position calculated by the ACU.
- **Acu_getIsMotorSpeedValid** to get the validity state of the motor speed calculated by the ACU.
- **Acu_getIsSensorElecPosValid** to get the validity state of the electrical position provided by a sensor to an ACU.
- **Acu_getMechPosition** to get the current mechanical position calculated by the ACU.
- **Acu_getMotorSpeed** to get the current motor speed calculated by the ACU.
- **Acu_getSensorElecPosition** to get the electrical position provided by a sensor to an ACU.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_apply	495
Acu_create	496
Acu_getCurrentInputSensorNo	501
Acu_getElecPosition	502
Acu_getIsElecPosValid	503
Acu_getIsMechPosValid	504
Acu_getIsMotorSpeedValid	505
Acu_getIsSensorElecPosValid	506

Acu_getMechPosition.....	508
Acu_getMotorSpeed.....	509
Acu_getSensorElecPosition.....	510
Acu_start.....	533

Acu_setAbsolutePositionForEvent

Syntax

```
UInt32 Acu_setAbsolutePositionForEvent(
    AcuSDrvObject *pAcu,
    UInt32 EventSelector,
    Float64 AbsoluteAnglePosition)
```

Include file

IoDrvAcu.h

Purpose

To specify an absolute angle position for the event generation of an ACU.

Description

The function is used to specify absolute positions at which events are generated. You can then specify which event to generate at a certain position by using **Acu_setInterruptMode** and **Acu_setTriggerMode**.

If the rotor position reaches the specified absolute position, the specified event is generated. You can define up to four different positions per ACU for the event generation by separate calls of **Acu_setAbsolutePositionForEvent**.

By default, the event-generating position to be specified refers to the electrical position. To set the mechanical position as the base for the angle position, use **Acu_setPositionEventBase**.

No events are generated at invalid positions.

The setting does not take effect until you call **Acu_apply** during the initialization phase or **Acu_write** during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

EventSelector Lets you select the absolute position event to be generated in the range 1 ... 4.

Symbol	Meaning
ACU_ABS_POSITION_EVENT_1	Specifies to generate event number 1 when the absolute position is reached.
...	...
ACU_ABS_POSITION_EVENT_4	Specifies to generate event number 4 when the absolute position is reached.

AbsoluteAnglePosition Lets you specify the absolute angle position in degrees in the range 0° ... 359.999999° with a resolution of 0.00000134°.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_apply.....	495
Acu_create.....	496

Acu_setInputSensor

Syntax

```
UInt32 Acu_setInputSensor(
    AcuSDrvObject *pAcu,
    UInt32 AcuSensorInputSelector,
    UInt32 InputSensorType,
    UInt32 InputSensorInstance)
```

Include file

IoDrvAcu.h

Purpose

To attach a position-generating sensor to an ACU.

Description	<p>The angle computation unit processes position values that are provided by up to two position-generating sensors (<i>Sensor A</i> and <i>Sensor B</i>). You must call Acu_setInputSensor for each sensor separately. The following types of sensors are available:</p> <ul style="list-style-type: none"> ▪ Incremental encoders ▪ Absolute encoders connected via EnDat or SSI interface ▪ Hall sensors ▪ Resolvers <p>If two sensors are attached to the ACU and only one of them provides valid position values, the ACU uses the valid values.</p> <p>If both sensors provide valid angle positions, <i>Sensor B</i> is used. Therefore, it is recommended to attach the sensor with the higher resolution as <i>Sensor B</i>.</p> <p>You can verify which of the two sensors is currently used by the ACU via Acu_getCurrentInputSensorNo.</p> <p>The setting does not take effect until you call Acu_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>
--------------------	--

Parameters	<p>pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using Acu_create.</p> <p>AcuSensorInputSelector Lets you specify whether the sensor is to be attached as <i>Sensor A</i> or <i>Sensor B</i>.</p>
-------------------	---

Symbol	Meaning
ACU_SENSOR_A	Attaches the specified sensor as <i>Sensor A</i> .
ACU_SENSOR_B	Attaches the specified sensor as <i>Sensor B</i> .

InputSensorType Lets you specify the type of the position-generating sensor that is to be attached.

Symbol	Meaning
ACU_SENSOR_NONE	No sensor is attached (default).
ACU_SENSOR_ENCODER	Specifies an incremental encoder.
ACU_SENSOR_HALL	Specifies a Hall sensor.
ACU_SENSOR_SSI	Specifies a sensor that is connected via SSI.
ACU_SENSOR_ENDAT	Specifies a sensor that is connected via EnDat.
ACU_SENSOR_RESOLVER	Specifies a resolver.

InputSensorInstance Lets you specify the instance number of the position-generating sensor to be attached to the ACU. The instance number is specified when the sensor's I/O driver object is created via an **xxx_create** function.

Symbol	Meaning
ACU_SENSOR_NONE	No sensor is attached.
ACU_ENCODER_INSTANCE_1	Specifies incremental encoder number 1 to be attached.
...	...
ACU_ENCODER_INSTANCE_6	Specifies incremental encoder number 6 to be attached.
ACU_HALL_INSTANCE_1	Specifies Hall sensor number 1 to be attached.
ACU_HALL_INSTANCE_2	Specifies Hall sensor number 2 to be attached.
ACU_ENDAT_INSTANCE_1	Specifies EnDat-connected sensor number 1 to be attached.
ACU_ENDAT_INSTANCE_2	Specifies EnDat-connected sensor number 2 to be attached.
ACU_SSI_INSTANCE_1	Specifies SSI-connected sensor number 1 to be attached.
ACU_SSI_INSTANCE_2	Specifies SSI-connected sensor number 2 to be attached.
ACU_RESOLVER_INSTANCE_1	Specifies resolver number 1 to be attached.
ACU_RESOLVER_INSTANCE_2	Specifies resolver number 2 to be attached.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply.....	495
Acu_create.....	496

Acu_setInterruptMode

Syntax

```
UInt32 Acu_setInterruptMode(
    AcuSDrvObject *pAcu,
    UInt32 InterruptMode)
```

Include file

IoDrvAcu.h

Purpose

To specify the interrupt mode for an ACU.

Description

This function is used to generate interrupts at the position values that are calculated by the specified ACU.

You can use the interrupt mode to select one or more previously defined positions at which to generate interrupts:

- Interrupt generation when the rotor reaches one of the absolute angle positions 1 ... 4 that you specified before via **Acu_setAbsolutePositionForEvent**.
- Interrupt generation when the rotor reaches one of the relative angle positions that you specified before via **Acu_setRelativePositionForEvent** and **Acu_setRelativePositionOffset**.

If you want to configure a combination of interrupts and trigger signals, you must configure the trigger mode in addition to the interrupt mode by using **Acu_setTriggerMode**.

The setting does not take effect until you call **Acu_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

InterruptMode Lets you specify the interrupt mode. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
ACU_NO_INTERRUPT	No interrupt is generated (default).
ACU_INT_ABSOLUTE_POSITION_1	Specifies to generate an interrupt every time the rotor reaches absolute position 1.
...	...
ACU_INT_ABSOLUTE_POSITION_4	Specifies to generate an interrupt every time the rotor reaches absolute position 4.
ACU_INT_RELATIVE_POSITION	Specifies to generate an interrupt every time the rotor reaches a relative position.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply	495
Acu_create	496
Acu_setPositionEventBase	521

Acu_setIoIntVector

Syntax

```
UInt32 Acu_setIoIntVector(
    AcuSDrvObject *pAcu,
    UInt32 SourceSelect,
    IoLibTToIntHandler IoIntHandler)
```

Include file

IoDrvAcu.h

Purpose

To set the interrupt handler for interrupts from an ACU.

Description

This function is used to register the function that handles the generated interrupts specified by using **Acu_setInterruptMode**. For each specified condition for interrupt generation (reaching the absolute or relative position), you can define separate functions. If you want to handle all conditions with the same function, you can use the bitwise OR operator to combine the predefined symbols of the **SourceSelect** parameter.

Note

The condition specified in the parameter **SourceSelect** must be a part of the conditions that are specified for the **Acu_setInterruptMode** function.

The setting does not take effect until you call **Acu_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

SourceSelect Lets you specify the position the given interrupt handler is to be associated with. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
ACU_INT_ABSOLUTE_POSITION_1	Associates the interrupt handler with interrupts generated at absolute position 1.
...	...
ACU_INT_ABSOLUTE_POSITION_4	Associates the interrupt handler with interrupts generated at absolute position 4.
ACU_INT_RELATIVE_POSITION	Associates the interrupt handler with interrupts generated at each of the relative positions.

IoIntHandler Lets you specify the address of the function which is to handle the generated interrupt.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_apply.....	495
Acu_create.....	496

Acu_setMinMotorSpeed

Syntax

```
UInt32 Acu_setMinMotorSpeed(
    AcuSDrvObject *pAcu,
    Float64 MinMotorSpeed)
```

Include file

IoDrvAcu.h

Purpose

To specify the minimal motor speed.

Description

The minimal motor speed is relevant for speed measurement and standstill detection. When the motor speed falls below this specified value, it is set to 0.

The setting does not take effect until you call **Acu_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

MinMotorSpeed Lets you specify the minimal motor speed in the range 1 rpm ... 1000 rpm.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_apply.....	495
Acu_create.....	496

Acu_setMotorPolePairCount

Syntax

```
UInt32 Acu_setMotorPolePairCount(
    AcuSDrvObject *pAcu,
    UInt32 NumberOfPolePairs)
```

Include file

IoDrvAcu.h

Purpose

To specify the motor's number of pole pairs.

Description

The number of pole pairs is required to evaluate the electrical position of the rotor. For example, it defines how many electrical rotations of the rotor correspond to one mechanical rotation.

The setting does not take effect until you call **Acu_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Note

The sensor and the motor must fulfill one of the following conditions:

- They have the same number of pole pairs.
- The motor's number of pole pairs is an integer multiple of the sensor's number of pole pairs.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

NumberOfPolePairs Lets you specify the number of pole pairs of the motor in the range 1 ... 16.

Symbol	Meaning
ACU_MOTOR_POLE_PAIRS_1	Specifies a motor with one pole pair (default).
...	...
ACU_MOTOR_POLE_PAIRS_16	Specifies a motor with 16 pole pairs.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_apply.....	495
Acu_create.....	496

Acu_setPositionEventBase

Syntax

```
UInt32 Acu_setPositionEventBase(
    AcuSDrvObject *pAcu,
    UInt32 ElectricalMechanicalSelection)
```

Include file	IoDrvAcu.h				
Purpose	To select the position type to be used for event generation of an ACU.				
Description	<p>This function is used to select one of the position values provided by the angle computation unit (ACU) to be used for event generation:</p> <ul style="list-style-type: none"> ▪ The electrical rotor position ▪ The mechanical rotor position <p>You can specify the rotor position angles at which events are generated by using Acu_setRelativePositionForEvent and Acu_setAbsolutePositionForEvent.</p> <p>The setting does not take effect until you call Acu_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>				
Parameters	<p>pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using Acu_create.</p> <p>ElectricalMechanicalSelection Lets you specify which of the ACU's position values to use for event generation.</p>				
Symbol	Meaning				
ACU_USE ELECTRICAL POSITION	Specifies the electrical position (default).				
ACU_USE MECHANICAL POSITION	Specifies the mechanical position.				
Return value	The function returns an error code.				
Error Code	Meaning				
IOLIB_NO_ERROR	The function was successfully completed.				
!= IOLIB_NO_ERROR	The function was not completed.				
Related topics	References				
	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px; width: 80%;">Acu_apply.....</td><td style="padding: 5px;">495</td></tr> <tr> <td style="padding: 5px;">Acu_create.....</td><td style="padding: 5px;">496</td></tr> </table>	Acu_apply.....	495	Acu_create.....	496
Acu_apply.....	495				
Acu_create.....	496				

Acu_setPositionEventHysteresis

Syntax

```
UInt32 Acu_setPositionEventHysteresis(
    AcuSDrvObject *pAcu,
    Float64 PositionEventHysteresis)
```

Include file

`IoDrvAcu.h`

Purpose

To set the hysteresis of the position that is relevant for the event generation of an ACU.

Description

This function prevents the generation of events in the stopped state. The measured position might oscillate around its standstill position during the stopped state due to the mechanical vibrations of a motor or due to the electrical noise during the evaluation of an analog position sensor. This might cause an undesirable generation of events.

You can use `Acu_setRelativePositionForEvent` and `Acu_setAbsolutePositionForEvent` to specify the rotor position angles to generate events.

The setting does not take effect until you call `Acu_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using `Acu_create`.

PositionEventHysteresis Lets you specify the hysteresis in the range 0.0° ... 60.0°.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics**References**

Acu_apply.....	495
Acu_create.....	496
Acu_setAbsolutePositionForEvent.....	513
Acu_setRelativePositionForEvent.....	524

Acu_setRelativePositionForEvent

Syntax

```
UInt32 Acu_setRelativePositionForEvent(
    AcuSDrvObject *pAcu,
    Float64 RelativeAnglePosition)
```

Include file

IoDrvAcu.h

Purpose

To specify the relative angle for periodic event generation of an ACU.

Description

Every time the rotor turns for the specified angle an event is generated. For example, if you specify a **RelativeAnglePosition** of 120°, an event will be generated at the rotor positions 0°, 120° and 240°.

The value of the relative angle position must be an integer divisor of 360° to achieve event generation at the same positions on every revolution of the rotor. You can specify an offset for these positions via **Acu_setRelativePositionOffset**.

By default, the event-generating position to be specified refers to the electrical position. To set the mechanical position as the base for the angle position, use **Acu_setPositionEventBase**.

The setting does not take effect until you call **Acu_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

RelativeAnglePosition Lets you specify the relative angle position in degrees in the range $0.00000134^\circ \dots 360^\circ$ with a resolution of 0.00000134° . If the specified value is not an integer divisor of 360° , it will be adjusted to the nearest integer divisor.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
\neq IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply	495
Acu_create	496
Acu_setPositionEventBase	521

Acu_setRelativePositionOffset

Syntax

```
UInt32 Acu_setRelativePositionOffset(
    AcuSDrvObject *pAcu,
    Float64 RelativePositionOffset)
```

Include file

IoDrvAcu.h

Purpose

To specify an offset angle for periodic event generation of an ACU.

Description

This function is used to specify an offset to the relative angle position for periodic event generation that you specify via **Acu_setRelativePositionForEvent**.

For example, if you specified a relative angle position of 120° and specify a **RelativePositionOffset** of 5° , an event will be generated at the rotor positions 5° , 125° and 245° .

The relative position offset value must be smaller than the relative angle position.

The setting does not take effect until you call **Acu_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters	<p>pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using Acu_create.</p> <p>RelativePositionOffset Lets you specify the relative position offset in degrees in the range $0.0^\circ \dots <\text{RelativePositionOffset} - 0.00000134>$ with a resolution of 0.00000134°.</p>
Return value	The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
\neq IOLIB_NO_ERROR	The function was not completed.

Related topics	References						
<table> <tr> <td>Acu_apply.....</td> <td>495</td> </tr> <tr> <td>Acu_create.....</td> <td>496</td> </tr> <tr> <td>Acu_setPositionEventBase.....</td> <td>521</td> </tr> </table>		Acu_apply	495	Acu_create	496	Acu_setPositionEventBase	521
Acu_apply	495						
Acu_create	496						
Acu_setPositionEventBase	521						

Acu_setSensorForPositionEvent

Syntax	<pre>UInt32 Acu_setSensorForPositionEvent(AcuSDrvObject *pAcu, UInt32 AcuSensorSelector)</pre>
Include file	<code>IoDrvAcu.h</code>
Purpose	To select the position-generating sensor as the base for event generation of an ACU.
Description	<p>You can select one of the position-generating sensors that are attached to the angle computation unit and designated as <i>Sensor A</i> and <i>Sensor B</i>.</p> <p>The setting does not take effect until you call Acu_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using [Acu_create](#).

AcuSensorSelector Lets you specify the position-generating sensor to be used for event generation.

Symbol	Meaning
ACU_SENSOR_RETAIN	Specifies to leave the current setting unchanged (default).
ACU_SENSOR_A	Specifies to use <i>Sensor A</i> for event generation.
ACU_SENSOR_B	Specifies to use <i>Sensor B</i> for event generation.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_apply	495
Acu_create	496

Acu_setSensorPolePairCount

Syntax

```
UInt32 Acu_setSensorPolePairCount(
    AcuSDrvObject *pAcu,
    UInt32 AcuSensorSelector,
    UInt32 NumberOfPolePairs)
```

Include file

IoDrvAcu.h

Purpose

To specify the sensor's number of pole pairs.

Description

The number of pole pairs is required to evaluate the electrical and the mechanical position of the rotor. It defines how many repetitions of a sensor's signal patterns correspond to one mechanical rotation of the rotor.

The setting does not take effect until you call **Acu_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Note

- The sensor and the motor must fulfill one of the following conditions:
 - They have the same number of pole pairs.
 - The motor's number of pole pairs is an integer multiple of the sensor's number of pole pairs.
- It makes sense to specify the number of pole pairs only for Hall sensors or resolvers. By default, the number of pole pairs for incremental encoders is one.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

AcuSensorSelector Lets you specify the position-generating sensor of the ACU to set the pole pair count for.

Symbol	Meaning
ACU_SENSOR_NONE	No sensor is selected (default).
ACU_SENSOR_A	Sets the number of pole pairs for Sensor A.
ACU_SENSOR_B	Sets the number of pole pairs for Sensor B.

NumberOfPolePairs Lets you specify the number of pole pairs of the sensor in the range 1 ... 16.

Symbol	Meaning
ACU_SENSOR_POLE_PAIRS_1	Specifies a sensor with one pole pair (default).
...	...
ACU_SENSOR_POLE_PAIRS_16	Specifies a sensor with 16 pole pairs.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_apply	495
Acu_create	496

Acu_setTriggerLineOut

Syntax

```
UInt32 Acu_setTriggerLineOut(
    AcuSDrvObject *pAcu,
    UInt32 TriggerLineOut)
```

Include file

IoDrvAcu.h

Purpose

To select the trigger line used for trigger event generation of an ACU.

Description

A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board. You must specify the trigger line the consumer must listen to by using the relevant `xxx_setTriggerLineIn` function. A trigger line can be allocated only once.

Before you can use trigger signals, you must set the trigger mode via `Acu_setTriggerMode`. With `Acu_setTriggerLineOutStatus` you enable or disable the trigger line.

The setting does not take effect until you call `Acu_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using `Acu_create`.

TriggerLineOut Lets you specify the number of the trigger line to be used for transmitting the trigger signal in the range 1 ... 16.

Symbol	Meaning
ACU_TRIGGER_LINE_1	Specifies trigger line 1.
...	...
ACU_TRIGGER_LINE_16	Specifies trigger line 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply.....	495
Acu_create.....	496
Acu_setTriggerLineOutStatus.....	530

Acu_setTriggerLineOutStatus

Syntax

```
UInt32 Acu_setTriggerLineOutStatus(
    AcuSDrvObject *pAcu,
    UInt32 Status)
```

Include file

IoDrvAcu.h

Purpose

To set the status of the trigger line.

Description

With this function, you can enable or disable the trigger line that you specified before by using **Acu_setTriggerLineOut**.

The function is intended to be called during the initialization phase of the real-time application or during the run time of the application.

The setting does not take effect until you call **Acu_start** during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using [Acu_create](#).

Status Lets you specify the status of the trigger line.

Symbol	Meaning
ACU_TRIGGER_LINE_DISABLE	Disables the trigger line.
ACU_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_create.....	496
Acu_setTriggerLineOut.....	529
Acu_start.....	533

Acu_setTriggerMode

Syntax

```
UInt32 Acu_setTriggerMode(
    AcuSDrvObject *pAcu,
    UInt32 TriggerMode)
```

Include file

IoDrvAcu.h

Purpose

To specify the trigger mode for an ACU.

Description

This function is used to generate trigger signals related to the position value that is calculated by the specified ACU.

Via the trigger mode you can select one or more previously defined positions where trigger signals are generated. The following trigger modes are available:

- Trigger signal generation when the rotor reaches one of the absolute positions 1 ... 4 that you specified before via **Acu_setAbsolutePositionForEvent**.
- Trigger signal generation when the rotor reaches one of the relative positions that you specified before via **Acu_setRelativePositionForEvent** and **Acu_setRelativePositionOffset**.

Additionally, you must specify the trigger line for the generated trigger signal via **Acu_setTriggerLineOut**. The I/O unit that is to consume the trigger signal has to specify this trigger line by using its related **xxx_setTriggerLineIn** function.

If you want to configure a combination of trigger signals and interrupts, you must configure the interrupt mode in addition to the trigger mode by using **Acu_setInterruptMode**.

The setting does not take effect until you call **Acu_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

TriggerMode Lets you specify the trigger mode.

Symbol	Meaning
ACU_NO_TRIGGER	No trigger is generated (default).
ACU_TRIG_ABSOLUTE_POSITION_1	Specifies to generate a trigger signal every time the rotor reaches absolute position 1.
...	...
ACU_TRIG_ABSOLUTE_POSITION_4	Specifies to generate a trigger signal every time the rotor reaches absolute position 4.
ACU_TRIG_RELATIVE_POSITION	Specifies to generate a trigger signal every time the rotor reaches a relative position.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

Acu_apply	495
Acu_create	496
Acu_setPositionEventBase	521

Acu_start

Syntax

```
UInt32 Acu_start(AcuSDrvObject *pAcu)
```

Include file

IoDrvAcu.h

Purpose

To start an angle computation unit.

Description

This function is used to start the specified ACU driver so that it activates its assigned angle computation unit. The data captured by the attached position-generating sensors is processed to provide the calculated electrical and mechanical position and the motor speed. This function also enables the configured trigger line. If you call **Acu_setTriggerLineOutStatus(ACU_TRIGGER_LINE_DISABLE)** before, the trigger line stays disabled.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Note

All angle computation units (ACUs) are disabled after reset. To re-enable an ACU, you must use **ACU_start**.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_apply.....	495
Acu_create.....	496
Acu_read.....	511
Acu_write.....	534

Acu_write

Syntax

```
UInt32 Acu_write(AcuSDrvObject *pAcu)
```

Include file

IoDrvAcu.h

Purpose

To update the settings of an angle computation unit during run time.

Description

If you used one of the following functions during run time, their settings will not be updated on the hardware until you call **Acu_write**:

- **Acu_setAbsolutePositionForEvent**

The specified ACU had to be enabled before by using **Acu_start**.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

Parameters

pAcu Lets you specify the angle computation unit to be used via the corresponding driver object. The ACU driver object had to be created before by using **Acu_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_create	496
Acu_disableInterrupts	498
Acu_enableInterrupts	499
Acu_setAbsolutePositionForEvent	513
Acu_start	533

EnDat Interface

Introduction

With the `DioCl2EnDatIn` functions, you can access absolute encoders via the EnDat interface on the DIO Class 2 channels to measure the position of a motor's rotor.

Where to go from here

Information in this section

<code>DioCl2EnDatIn_apply</code>	537
To apply the initialization settings to the EnDat interface.	
<code>DioCl2EnDatIn_create</code>	538
To create the I/O driver object for operating an absolute encoder on the EnDat interface via DIO Class 2 input channels.	
<code>DioCl2EnDatIn_getConfigErr</code>	540
To check the consistency between the configuration of the EnDat interface and the parameters of the connected encoder.	
<code>DioCl2EnDatIn_getCrcErrorCount</code>	542
To get the total number of transmission CRC errors.	
<code>DioCl2EnDatIn_getIsPositionValid</code>	543
To get the result of the position validity check of the EnDat interface.	
<code>DioCl2EnDatIn_getIsRevolutionNoValid</code>	544
To check the validity of the revolution number provided by the EnDat interface.	
<code>DioCl2EnDatIn_getMechPosition</code>	546
To get the current mechanical position measured via the EnDat interface.	
<code>DioCl2EnDatIn_getRevolutionNo</code>	548
To get the current number of revolutions measured via the EnDat interface.	
<code>DioCl2EnDatIn_getTransmissionErr</code>	549
To get information on transmission errors.	
<code>DioCl2EnDatIn_read</code>	551
To read data from the EnDat interface.	
<code>DioCl2EnDatIn_setClockFrequency</code>	552
To specify the frequency of the clock output signal of the EnDat interface.	
<code>DioCl2EnDatIn_setCrcErrorLimit</code>	555
To specify the number of CRC errors to be ignored before a CRC error becomes effective.	
<code>DioCl2EnDatIn_setMultiturnRes</code>	556
To specify the multi-turn resolution of an absolute encoder connected to the EnDat interface.	

DioCl2EnDatIn_setNumberOfSteps	558
To specify the number of steps of an absolute encoder connected to the EnDat interface.	
DioCl2EnDatIn_setPositionOffset	559
To specify an offset angle for the evaluated encoder position.	
DioCl2EnDatIn_setReverseDirection	560
To reverse the rotational direction in the EnDat interface.	
DioCl2EnDatIn_setSampleSyncEvSrc	562
To specify the event source triggering the data transmission to the EnDat interface.	
DioCl2EnDatIn_setSingleturnRes	563
To specify the single-turn resolution of an absolute encoder connected to the EnDat interface.	
DioCl2EnDatIn_start	565
To start generating and measuring signals via the EnDat interface.	
DioCl2EnDatIn_stop	566
To deactivate the EnDat interface.	

DioCl2EnDatIn_apply

Syntax

```
UInt32 DioCl2EnDatIn_apply(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn)
```

Include file

`IoDrvDioClass2EnDatIn.h`

Purpose

To apply the initialization settings to the EnDat interface.

Description

Before you can start the specified EnDat interface, you must transfer the settings of the DioCl2EnDatIn driver object to its related input and output channels by using **DioCl2EnDatIn_apply**.

You must do this also for the default values and for values that you specified via the **DioCl2EnDatIn_set<Parameter>** functions. The **DioCl2EnDatIn_apply** function is intended to be called at the end of the initialization phase of the real-time application.

The signal generation and measurement on the specified interface are not activated until you call **DioCl2EnDatIn_start**.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using [DioCl2EnDatIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_start.....	565

[DioCl2EnDatIn_create](#)**Syntax**

```
UInt32 DioCl2EnDatIn_create(
    DioCl2EnDatInSDrvObject **ppDioCl2EnDatIn,
    UInt32 EnDatInUnitNo,
    UInt32 Channel)
```

Include file

`IoDrvDioClass2EnDatIn.h`

Purpose

To create the I/O driver object for operating an absolute encoder on the EnDat interface via DIO Class 2 input channels.

Description

This function is used to perform all the steps necessary to create an I/O driver object to access an EnDat interface.

A DioCl2EnDatIn driver object handles one EnDat interface with one output channel for the clock signal (first channel) and one bidirectional channel for the data signal (second channel).

You allocate the required consecutive channels by specifying the first channel.

The I/O driver is initialized with default values so that it is ready for use.

The initial values are:

- Clock frequency: 1 MHz
- Encoder type: single-turn encoder
- Position offset: 0 degrees
- Reverse direction: do not reverse direction
- Single-turn resolution: 13 bit
- Multi-turn resolution: 0 bit
- Number of steps: DIO_CLASS2_ENDAT_DEFAULT_STEPS (corresponds to the single-turn resolution)
- Sample synchronization event: continuous
- CRC error limit: 1

The function is intended to be called during the initialization phase of the real-time application.

You can use the **DioCl2EnDatIn_set<Parameter>** functions to change the default parameters. Before you can start the specified EnDat interface, you have to transfer the settings of the DioCl2EnDatIn driver object to its related I/O channels by using **DioCl2EnDatIn_apply**.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

ppDioCl2EnDatIn Lets you specify the address of a variable which holds the address of the created driver object for the EnDat interface.

EnDatInUnitNo Lets you specify the instance number of the EnDat interface unit in the range 1 ... 2.

Symbol	Meaning
DIO_CLASS2_ENDAT_IN_UNIT_1	Specifies EnDat interface 1.
DIO_CLASS2_ENDAT_IN_UNIT_2	Specifies EnDat interface 2.

Channel Lets you specify the channel to be used for the clock output signal in the range 1 ... 11. The subsequent channel is used as the data channel.

These channels must not be allocated by other DIO Class 2 functions.

Symbol	Meaning
DIO_CLASS2_CHANNEL_1	Specifies channel 1 of DIO Class 2.
...	...
DIO_CLASS2_CHANNEL_11	Specifies channel 11 of DIO Class 2.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

[DioCl2EnDatIn_apply](#)..... 537

DioCl2EnDatIn_getConfigErr

Syntax

```
UInt32 DioCl2EnDatIn_getConfigErr(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,
    UInt32 *pConfigError)
```

Include file

`IoDrvDicClass2EnDatIn.h`

Purpose

To check the consistency between the configuration of the EnDat interface and the parameters of the connected encoder.

Description

To work properly, the configuration of the EnDat interface must match the connected encoder. The consistency check of the configuration requires communication with the connected encoder. You therefore have to call this function during the run time of your real-time application.

A configuration error information is available if an inconsistency occurred for one of the following encoder parameters:

- Number of steps
- Single-turn resolution

- Multi-turn resolution
- Clock frequency (only for EnDat 2.2 sensors)

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using [DioCl2EnDatIn_create](#).

pConfigError Lets you specify the address of a variable that holds the configuration error information.

The following predefined symbols are used. They can be combined via the logical OR operator.

Symbol	Meaning
DIO_CLASS2_ENDAT_CHECK_MISSING (0x01)	Indicates that the EnDat interface configuration could not be checked because the parameters cannot be read from the connected encoder.
DIO_CLASS2_ENDAT_STEP_COUNT_ERR (0x02)	Indicates that the specified number of steps in the EnDat interface is inconsistent with the connected encoder.
DIO_CLASS2_ENDAT_SINGLE_RES_ERR (0x04)	Indicates that the specified single-turn resolution in the EnDat interface is inconsistent with the connected encoder.
DIO_CLASS2_ENDAT_MULTI_RES_ERR (0x08)	Indicates that the specified multi-turn resolution in the EnDat interface is inconsistent with the connected encoder.
DIO_CLASS2_ENDAT_CLOCK_FREQ_ERR (0x10)	Indicates that the specified clock frequency in the EnDat interface is inconsistent with the connected encoder.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References						
	<table> <tr> <td>DioCl2EnDatIn_create.....</td> <td>538</td> </tr> <tr> <td>DioCl2EnDatIn_read.....</td> <td>551</td> </tr> <tr> <td>DioCl2EnDatIn_start.....</td> <td>565</td> </tr> </table>	DioCl2EnDatIn_create.....	538	DioCl2EnDatIn_read.....	551	DioCl2EnDatIn_start.....	565
DioCl2EnDatIn_create.....	538						
DioCl2EnDatIn_read.....	551						
DioCl2EnDatIn_start.....	565						

DioCl2EnDatIn_getCrcErrorCount

Syntax	<pre>UInt32 DioCl2EnDatIn_getCrcErrorCount(DioCl2EnDatInSDrvObject *pDioCl2EnDatIn, UInt32 *pCrcErrorCount)</pre>
Include file	IoDrvDioClass2EnDatIn.h
Purpose	To get the total number of transmission CRC errors.
Description	<p>The EnDat interface provides a counter for the CRC errors that occurred during transmission. The counter starts working when you start the real-time application.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call DioCl2EnDatIn_read to update the internal buffer with the latest values.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>
Parameters	<p>pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using DioCl2EnDatIn_create.</p> <p>pCrcErrorCount Lets you specify the address of a variable that holds the total number of transmission CRC errors in the range 0 ... 65535.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_getTransmissionErr.....	549
DioCl2EnDatIn_read.....	551

DioCl2EnDatIn_getIsPositionValid

Syntax

```
UInt32 DioCl2EnDatIn_getIsPositionValid(
    DioCl2EnDatInDrvObject *pDioCl2EnDatIn,
    UInt32 *pIsPositionValid)
```

Include file

IoDrvDioClass2EnDatIn.h

Purpose

To get the result of the position validity check of the EnDat interface.

Description

This function is used to indicate whether the measured position that you have read by using **DioCl2EnDatIn_getMechPosition** is valid.

The validity check includes the following information:

- Status of the error bit F1 contained in the EnDat protocol
- Status of the CRC checksum contained in the EnDat protocol
- Physical connection to an absolute encoder
- Availability and correctness of the encoder configuration in the EnDat interface

A connection error, a configuration error, or the error bit F1 always results in an invalid position value. For CRC errors, you can specify a tolerance by using **DioCl2EnDatIn_setCrcErrorLimit**.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl2EnDatIn_read** to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#)
[\(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using [DioCl2EnDatIn_create](#).

pIsPositionValid Lets you specify the address of a variable that holds the result of the position validity check.

Result Value	Meaning
0	The measured position value is not valid.
>0	The measured position value is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_getMechPosition.....	546
DioCl2EnDatIn_read.....	551
DioCl2EnDatIn_setCrcErrorLimit.....	555

DioCl2EnDatIn_getIsRevoltNoValid

Syntax

```
UInt32 DioCl2EnDatIn_getIsRevoltNoValid(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,
    UInt32 *pIsRevoltNoValid)
```

Include file

[IoDrvDioClass2EnDatIn.h](#)

Purpose To check the validity of the revolution number provided by the EnDat interface.

Description This function is used to indicate whether the measured revolution number that you have read by using `DioCl2EnDatIn_getRevolutionNo` is valid.

The validity check includes the following information:

- Status of the error bit F1 contained in the EnDat protocol
- Status of the CRC checksum contained in the EnDat protocol
- Type of the connected encoder (single-turn or multi-turn encoder)
- Physical connection to an absolute encoder
- Availability and correctness of the encoder configuration in the EnDat interface

If a single-turn encoder is connected, a measured revolution number is always invalid. Only multi-turn encoders count the revolutions.

A connection error, a configuration error, or the error bit F1 always results in an invalid revolution number. For CRC errors, you can specify a tolerance by using `DioCl2EnDatIn_setCrcErrorLimit`.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call `DioCl2EnDatIn_read` to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters **pDioCl2EnDatIn** Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using `DioCl2EnDatIn_create`.

plsRevolutionNoValid Lets you specify the address of a variable that holds the result of the revolution number validity check.

Result Value	Meaning
0	The measured revolution number is not valid.
>0	The measured revolution number is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_getRevolutionNo.....	548
DioCl2EnDatIn_read.....	551
DioCl2EnDatIn_setCrcErrorLimit.....	555

DioCl2EnDatIn_getMechPosition

Syntax

```
UInt32 DioCl2EnDatIn_getMechPosition(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,
    Float64 *pMechPosition)
```

Include file

IoDrvDioClass2EnDatIn.h

Purpose

To get the current mechanical position measured via the EnDat interface.

Description

This function is used to provide the current mechanical position that is evaluated by the EnDat interface. The angle position value includes the optional offset specified by **DioCl2EnDatIn_setPositionOffset**.

The resolution of the position is (360/NumberOfSteps) degrees. The number of steps is specified by **DioCl2EnDatIn_setNumberOfSteps**. Use the **DioCl2EnDatIn_getIsPositionValid** function to validate the measured position value.

Note

All channels of the DIO Class 2 unit are in a high impedance state after reset. To enable the clock output of the absolute encoder that you specified by using **DioCl2EnDatIn_create**, you must use **DioCl2EnDatIn_start**.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl2EnDatIn_read** to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

Note

If you use an incremental encoder via the EnDat interface, this function will not output the mechanical (absolute) angle position but the relative angle position determined by the incremental encoder. Usually, the starting point for the relative angle position is set when the incremental encoder is powered.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using [DioCl2EnDatIn_create](#).

pMechPosition Lets you specify the address of a variable that holds the current mechanical position of the absolute encoder in degrees in the range $0^\circ \dots (360.0 \cdot (360/\text{NumberOfSteps}))^\circ$ with a resolution of $(360/\text{NumberOfSteps})^\circ$.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
\neq IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_getsPositionValid.....	543
DioCl2EnDatIn_read.....	551
DioCl2EnDatIn_setNumberOfSteps.....	558
DioCl2EnDatIn_setPositionOffset.....	559

DioCl2EnDatIn_getRevolutionNo

Syntax	<pre>UInt32 DioCl2EnDatIn_getRevolutionNo(DioCl2EnDatInSDrvObject *pDioCl2EnDatIn, UInt32 *pRevolutionNo)</pre>
Include file	<code>IoDrvDioClass2EnDatIn.h</code>
Purpose	To get the current number of revolutions measured via the EnDat interface.
Description	<p>This function is used to provide the current number of revolutions that is evaluated by the EnDat interface. This value is available only when you use a multi-turn encoder. The revolution number starts with 0. The maximum number of revolutions depends on the multi-turn resolution that you specified by using <code>DioCl2EnDatIn_setMultiturnRes</code>.</p> <p>Note</p> <p>All channels of the DIO Class 2 unit are in a high impedance state after reset. To enable the clock output of the absolute encoder that you specified by using <code>DioCl2EnDatIn_create</code>, you must use <code>DioCl2EnDatIn_start</code>.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call <code>DioCl2EnDatIn_read</code> to update the internal buffer with the latest values.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>
Parameters	<p>pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using <code>DioCl2EnDatIn_create</code>.</p> <p>pRevolutionNo Lets you specify the address of a variable that holds the number of the current revolution.</p> <p>The maximum value is $2^{28}-1$.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_getIsRevoltValid.....	544
DioCl2EnDatIn_read.....	551
DioCl2EnDatIn_setMultiTurnRes.....	556
DioCl2EnDatIn_start.....	565

DioCl2EnDatIn_getTransmissionErr

Syntax

```
UInt32 DioCl2EnDatIn_getTransmissionErr(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,
    UInt32 *pTransmissionError)
```

Include file

IoDrvDioClass2EnDatIn.h

Purpose

To get information on transmission errors.

Description

A transmission error information is available if at least one of the following errors occurred at the EnDat interface:

- Error bit F1 was set in the EnDat protocol
- CRC checksum error occurred for corrupted data (F1 bit, position data, and the CRC checksum itself) in the EnDat protocol
- Connection error, if no absolute encoder is connected to the EnDat interface or the connected encoder is not responding
- Parameter read error, if reading the EnDat parameters from the connected encoder failed

A connection error, a configuration error, or the error bit F1 always results in an error information. For CRC errors, you can specify a tolerance by using [DioCl2EnDatIn_setCrcErrorLimit](#).

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call `DioCl2EnDatIn_read` to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using `DioCl2EnDatIn_create`.

pTransmissionError Lets you specify the address of a variable that holds the transmission error information.

The following predefined symbols are used. They might be combined via the logical OR operator.

Symbol	Meaning
DIO_CLASS2_ENDAT_PAR_READ_ERROR (0x01)	Indicates that an error occurred when reading the EnDat parameters from the encoder.
DIO_CLASS2_ENDAT_CRC_ERROR (0x02)	Indicates that a CRC error was detected in the EnDat transmission.
DIO_CLASS2_ENDAT_ERROR_F1_SET (0x04)	Indicates that an error bit F1 was set in the EnDat transmission.
DIO_CLASS2_ENDAT_NO_CONNECTION (0x08)	Indicates that no encoder is connected to the EnDat interface, or that the connected encoder does not respond.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_read.....	551

DioCl2EnDatIn_setCrcErrorLimit..... 555

DioCl2EnDatIn_read

Syntax

```
UInt32 DioCl2EnDatIn_read(  
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn)
```

Include file

IoDrvDioClass2EnDatIn.h

Purpose

To read data from the EnDat interface.

Description

This function reads the data from the absolute encoder connected to the EnDat interface, checks the EnDat interface status (configuration and transmission error information), evaluates the validity of the position and the revolution number, and stores the results as a consistent data set in an internal buffer.

If you specified an offset by using **DioCl2EnDatIn_setPositionOffset**, it is included in the read angle position.

The input and output signals must be enabled before via **DioCl2EnDatIn_start**.

To get the values from the internal buffer, you can use one of the following functions:

- **DioCl2EnDatIn_getMechPosition** to get the value of the mechanical position
- **DioCl2EnDatIn_getRevolutionNo** to get the number of the current revolution
- **DioCl2EnDatIn_getIsPositionValid** to get the information on whether the measured position value is valid
- **DioCl2EnDatIn_getIsRevolNoValid** to get the information on whether the measured revolution number is valid
- **DioCl2EnDatIn_getTransmissionErr** to get the information on transmission errors
- **DioCl2EnDatIn_getConfigErr** to get the information on configuration errors
- **DioCl2EnDatIn_getCrcErrorCount** to get the number of CRC errors

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) .
	For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features) .

Parameters	pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using DioCl2EnDatIn_create .
-------------------	---

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References														
	<table border="0"> <tr> <td>DioCl2EnDatIn_create.....</td> <td>538</td> </tr> <tr> <td>DioCl2EnDatIn_getConfigErr.....</td> <td>540</td> </tr> <tr> <td>DioCl2EnDatIn_getMechPosition.....</td> <td>546</td> </tr> <tr> <td>DioCl2EnDatIn_getRevolutionNo.....</td> <td>548</td> </tr> <tr> <td>DioCl2EnDatIn_getTransmissionErr.....</td> <td>549</td> </tr> <tr> <td>DioCl2EnDatIn_setPositionOffset.....</td> <td>559</td> </tr> <tr> <td>DioCl2EnDatIn_start.....</td> <td>565</td> </tr> </table>	DioCl2EnDatIn_create.....	538	DioCl2EnDatIn_getConfigErr.....	540	DioCl2EnDatIn_getMechPosition.....	546	DioCl2EnDatIn_getRevolutionNo.....	548	DioCl2EnDatIn_getTransmissionErr.....	549	DioCl2EnDatIn_setPositionOffset.....	559	DioCl2EnDatIn_start.....	565
DioCl2EnDatIn_create.....	538														
DioCl2EnDatIn_getConfigErr.....	540														
DioCl2EnDatIn_getMechPosition.....	546														
DioCl2EnDatIn_getRevolutionNo.....	548														
DioCl2EnDatIn_getTransmissionErr.....	549														
DioCl2EnDatIn_setPositionOffset.....	559														
DioCl2EnDatIn_start.....	565														

DioCl2EnDatIn_setClockFrequency

Syntax	<pre>UInt32 DioCl2EnDatIn_setClockFrequency(DioCl2EnDatInSDrvObject *pDioCl2EnDatIn, Float64 ClockFrequency)</pre>
---------------	---

Include file	<code>IoDrvDioClass2EnDatIn.h</code>
---------------------	--------------------------------------

Purpose	To specify the frequency of the clock output signal of the EnDat interface.
----------------	---

Description

Note the following restrictions on the configuration of the clock signal frequency:

- The EnDat interface is only able to generate clock frequencies that are derived from 100 MHz by dividing them by an integer value ($f_{\text{effective}} = 100 \text{ MHz} / n$).
- For calculating the effective clock frequency, the two frequencies that result by the two neighboring integer values of the specified frequency are considered. This gives you one frequency that is equal to or lower than the specified frequency, and one value that is equal to or higher than the specified one. If the difference to the higher value exceeds 5%, the lower value is used. If the difference to the higher value is less than or equal to 5%, the value nearer to the specified value is used.

Example:

You have an absolute encoder that supports a clock frequency of 8 MHz.

```
fspecified = 8 MHz
n = 100 MHz / 8 MHz = 12.5
fhigher = 100 MHz / 12 = 8.33 MHz
    (has a deviation of 4.125% that is <5%)
flower = 100 MHz / 13 = 7.69 MHz
    (has a deviation of 3.875%)
```

In this example, the specified frequency results in an effective frequency of 7.69 MHz.

- According to the EnDat 2.2 specification, clock frequencies up to 8 MHz can be used with a maximum cable length of 100 m. At higher clock frequencies, the maximum cable length decreases to up to 20 m.

Clock Frequency	Maximum Cable Length
Up to 8.0 MHz	100 m
8.0 ... 8.5 MHz	90 m
8.5 ... 9.0 MHz	80 m
9.0 ... 10.5 MHz	60 m
10.5 ... 11.5 MHz	50 m
11.5 ... 12.5 MHz	40 m
12.5 ... 14.0 MHz	30 m
14.0 ... 16.0 MHz	20 m

- The default waiting time between two data transmissions (also called recovery time) is in the range 10 ... 30 μ s. If you specify a clock frequency of 1 MHz or higher for an EnDat 2.2 sensor, the sensor is automatically configured to the shorter waiting time of 1.25 ... 3.75 μ s. Otherwise, the waiting time might take longer than the time for data transmission. This behavior corresponds to the EnDat 2.2 specification.

Note

The configuration of the shorter waiting time is stored in the sensor's non-volatile memory. After its activation, EnDat 2.2 commands cannot be used with a clock frequency less than 1 MHz.

If you want to use the sensor with another hardware system with a clock frequency less than 1 MHz, you have to clear the shorter waiting time configuration. To do so, specify a clock frequency less than 1 MHz and start the real-time application with the connected sensor once.

The setting takes effect only after you call `DioCl2EnDatIn_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using `DioCl2EnDatIn_create`.

ClockFrequency Lets you specify the frequency of the signal generated for the clock output of the EnDat interface in Hz in the range 100 kHz ... 16 MHz. For clock frequencies greater than 8 MHz, you have to note, that the maximum cable length that can be used decreases to up to 20 m.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_apply	537
DioCl2EnDatIn_create	538

DioCl2EnDatIn_setCrcErrorLimit

Syntax

```
UInt32 DioCl2EnDatIn_setCrcErrorLimit(  
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,  
    UInt32 CrcErrorLimit)
```

Include file

IoDrvDicClass2EnDatIn.h

Purpose

To specify the number of CRC errors to be ignored before a CRC error becomes effective.

Description

The EnDat protocol provides a CRC value in each data transmission. With the CRC error limit, you can specify the number of successive CRC errors that are to be ignored before the EnDat interface raises an CRC error.

A CRC error can be detected by using [DioCl2EnDatIn_getTransmissionErr](#).

If a CRC error is raised, the current values for the position and the revolution number are not valid, refer to [DioCl2EnDatIn_getIsPositionValid](#) and [DioCl2EnDatIn_getIsRevolNoValid](#).

The setting takes effect only after you call [DioCl2EnDatIn_apply](#) during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using [DioCl2EnDatIn_create](#).

CrcErrorLimit Lets you specify the number of consecutive CRC errors to be ignored before a CRC error becomes effective in the range 0 ... 255.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_apply.....	537
DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_getCrcErrorCount.....	542
DioCl2EnDatIn_getIsPositionValid.....	543
DioCl2EnDatIn_getIsRevolutionValid.....	544
DioCl2EnDatIn_getTransmissionErr.....	549

DioCl2EnDatIn_setMultiturnRes

Syntax

```
UInt32 DioCl2EnDatIn_setMultiturnRes(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,
    UInt32 MultiturnResolution)
```

Include file

IoDrvDioClass2EnDatIn.h

Purpose

To specify the multi-turn resolution of an absolute encoder connected to the EnDat interface.

Description

The multi-turn resolution is a property of the connected encoder. It describes the number of bits that are used to provide the number of the current revolution. Implicitly, the multi-turn resolution defines the maximum number of revolutions the encoder is able to count.

The configuration of the EnDat interface depends on the type of the connected encoder.

- Using single-turn encoder

You have to specify the single-turn resolution by using **DioCl2EnDatIn_setSingleTurnRes**. If the number of steps is not equal to $2^{\text{SingleTurnResolution}}$, you have to explicitly specify the number of steps by using **DioCl2EnDatIn_setNumberOfSteps**. Optionally, you can specify the multi-turn resolution with 0.

- Using multi-turn encoder

You have to specify the single-turn resolution by using **DioCl2EnDatIn_setSingleturnRes** and the multi-turn resolution. The number of steps is then defined implicitly.

For further information on the encoder types, refer to [EnDat Interface \(MicroLabBox Features\)](#).

The setting takes effect only after you call **DioCl2EnDatIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using **DioCl2EnDatIn_create**.

MultiturnResolution Lets you specify the number of bits that the connected encoder is used to provide the current number of revolutions in the range 0 ... 28 bit.

If you specify 0 bit, the EnDat interface is configured for a single-turn encoder.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2EnDatIn_apply.....	537
DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_setNumberOfSteps.....	558
DioCl2EnDatIn_setSingleturnRes.....	563

DioCl2EnDatIn_setNumberOfSteps

Syntax

```
UInt32 DioCl2EnDatIn_setNumberOfSteps(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,
    UInt32 NumberOfSteps)
```

Include file

`IoDrvDioClass2EnDatIn.h`

Purpose

To specify the number of steps of an absolute encoder connected to the EnDat interface.

Description

The number of steps is a property of the connected encoder. It describes into how many sections the connected encoder divides one revolution.

The configuration of the EnDat interface depends on the type of the connected encoder.

- Using single-turn encoder

If the connected encoder supports a number of steps less than `2SingleturnResolution`, you have to explicitly specify the number of steps. If the number of steps provided by the encoder corresponds to its single-turn resolution, you do not need to call this function. You can also use the `DIO_CLASS2_ENDAT_DEFAULT_STEPS` define to achieve more flexibility in your source code.

If you have specified a value greater than `2SingleturnResolution`, the function exits with an error message.

- Using multi-turn encoder

The number of steps is implicitly given by the encoder's single-turn resolution. Calling `DioCl2_EnDatIn_setNumberOfSteps` is not required. If you use this function, you have to specify `2SingleturnResolution` or `DIO_CLASS2_ENDAT_DEFAULT_STEPS` for the number of steps. Otherwise, the function exits with an error message.

For further information on the encoder types, refer to [EnDat Interface \(MicroLabBox Features\)](#).

The setting takes effect only after you call `DioCl2EnDatIn_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters	<p>pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using DioCl2EnDatIn_create.</p> <p>NumberOfSteps Lets you specify the number of steps provided by the connected encoder in the range 1 ... 2^{28}. With the DIO_CLASS2_ENDAT_DEFAULT_STEPS define, you can specify to use the number of steps implicitly given by the encoder's single-turn resolution.</p>
Return value	The function returns an error code.

Related topics	References
	<p>DioCl2EnDatIn_apply..... 537 DioCl2EnDatIn_create..... 538 DioCl2EnDatIn_setSingleturnRes..... 563</p>

DioCl2EnDatIn_setPositionOffset

Syntax	<pre>UInt32 DioCl2EnDatIn_setPositionOffset(DioCl2EnDatInSDrvObject *pDioCl2EnDatIn, Float64 PositionOffset)</pre>
Include file	<code>IoDrvDioClass2EnDatIn.h</code>
Purpose	To specify an offset angle for the evaluated encoder position.
Description	<p>This function is used to specify an offset angle for the measured angular position of the absolute encoder. For example, the offset value lets you trigger the commutation of the motor earlier (positive offset) or later (negative offset) than the measured encoder position.</p> <p>The setting takes effect only after you call DioCl2EnDatIn_apply during the initialization phase.</p>

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using [DioCl2EnDatIn_create](#).

PositionOffset Lets you specify the position offset in degrees in the range -360.0° ... +360.0° with a resolution of 0.0055°.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_apply.....	537
DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_getMechPosition.....	546
DioCl2EnDatIn_read.....	551

DioCl2EnDatIn_setReverseDirection

Syntax

```
UInt32 DioCl2EnDatIn_setReverseDirection(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,
    UInt32 Reversing)
```

Include file

[IoDrvDioClass2EnDatIn.h](#)

Purpose

To reverse the rotational direction in the EnDat interface.

Description

The rotational direction that is given by an absolute encoder can be reversed in the EnDat interface that the encoder is connected to.

If there is no reversing, the forward rotation is assumed to be the clockwise rotation (from the front view of the motor shaft). For information on the default direction of the rotation, refer to the encoder's documentation. If you have activated reversing, a clockwise rotation will be measured as a backward rotation.

This function is useful if it is required to install the absolute encoder in inverse orientation.

The setting takes effect only after you call `DioCl2EnDatIn_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Note

All channels of the DIO Class 2 unit are in a high impedance state after reset. To enable the clock output of the absolute encoder that you specified by using `DioCl2EnDatIn_create`, you must use `DioCl2EnDatIn_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using `DioCl2EnDatIn_create`.

Reversing Lets you specify whether to reverse the encoder data in the EnDat interface.

Symbol	Meaning
DIO_CLASS2_ENDAT_REVERS_INACTIVE	Specifies to interpret increasing position values in the encoder data as forward rotation.
DIO_CLASS2_ENDAT_REVERS_ACTIVE	Specifies to interpret increasing position values in the encoder data as backward rotation.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_apply.....	537
DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_start.....	565

DioCl2EnDatIn_setSampleSyncEvSrc

Syntax

```
UInt32 DioCl2EnDatIn_setSampleSyncEvSrc(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,
    UInt32 SampleSyncEventSource)
```

Include file

IoDrvDioClass2EnDatIn.h

Purpose

To specify the event source triggering the data transmission to the EnDat interface.

Description

The EnDat interface is able to synchronize the transmission of the position information with other events in the real-time application that are transferred via a trigger line. For example, you can use rising edges of a digital signal as trigger event, or the detection of the middle of a period when using multichannel PWM signal generation.

If you do not specify a trigger line number that the EnDat interface is listening to, the data transmission is done continuously based on the clock frequency.

The setting takes effect only after you call **DioCl2EnDatIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using [DioCl2EnDatIn_create](#).

SampleSyncEventSource Lets you specify the number of the trigger line used as trigger source for data transmission via the EnDat interface in the range 1 ... 16. There is an additional symbol available to specify not to use a trigger line.

Symbol	Meaning
DIO_CLASS2_TRIGGER_LINE_NONE	No trigger line is used. The data transmission is processed continuously.
DIO_CLASS2_TRIGGER_LINE_1	Specifies trigger line 1.
...	...
DIO_CLASS2_TRIGGER_LINE_16	Specifies trigger line 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_apply.....	537
DioCl2EnDatIn_create.....	538

DioCl2EnDatIn_setSingletturnRes

Syntax

```
UInt32 DioCl2EnDatIn_setSingletturnRes(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn,
    UInt32 SingletturnResolution)
```

Include file

IoDrvDioClass2EnDatIn.h

Purpose	To specify the single-turn resolution of an absolute encoder connected to the EnDat interface.						
Description	<p>The single-turn resolution is a property of the connected encoder. It describes the number of bits that are used to provide the value of a specific angular position within one revolution.</p> <p>The configuration of the EnDat interface depends on the type of the connected encoder.</p> <ul style="list-style-type: none"> ▪ Using single-turn encoder <p>If the number of steps is not equal to $2^{\text{SingletturnResolution}}$, you have to explicitly specify the number of steps by using <code>DioCl2EnDatIn_setNumberOfSteps</code>. Optionally, you can specify the multi-turn resolution with 0 by using <code>DioCl2EnDatIn_setMultiturnRes</code>.</p> <ul style="list-style-type: none"> ▪ Using multi-turn encoder <p>You have to specify the single-turn resolution and the multi-turn resolution by using <code>DioCl2EnDatIn_setMultiturnRes</code>. The number of steps is then defined implicitly as $2^{\text{SingletturnResolution}}$.</p> <p>For further information on the encoder types, refer to EnDat Interface (MicroLabBox Features).</p> <p>The setting takes effect only after you call <code>DioCl2EnDatIn_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using <code>DioCl2EnDatIn_create</code>.</p> <p>SingletturnResolution Lets you specify the number of bits that the connected encoder is used to provide an angular position within one revolution in the range 1 ... 28 bit.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics

References

DioCl2EnDatIn_apply.....	537
DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_setMultiturnRes.....	556
DioCl2EnDatIn_setNumberOfSteps.....	558

DioCl2EnDatIn_start

Syntax

```
UInt32 DioCl2EnDatIn_start(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn)
```

Include file

IoDrvDioClass2EnDatIn.h

Purpose

To start generating and measuring signals via the EnDat interface.

Description

This function is used to start the specified EnDat interface so that it will activate its channels. The first digital channel is enabled to start generating the Clock signal. The second digital channel is enabled to transmit and receive the Data signal.

Since the EnDat interface is not started, the channels are in high impedance state.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using [DioCl2EnDatIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_apply.....	537
DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_read.....	551
DioCl2EnDatIn_stop.....	566

DioCl2EnDatIn_stop

Syntax

```
UInt32 DioCl2EnDatIn_stop(
    DioCl2EnDatInSDrvObject *pDioCl2EnDatIn)
```

Include file

`IoDrvDioClass2EnDatIn.h`

Purpose

To deactivate the EnDat interface.

Description

This function is used to stop the transmissiong of data via the specified EnDat interface. Using `DioCl2EnDatIn_read` will result in an error message.

When performing a stop/start transition for the EnDat interface, neither the EnDat interface nor the connected encoder will be reset, e.g., the CRC error counter continues with the last value when restarting the EnDat interface.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EnDatIn Lets you specify the DioCl2EnDatIn driver object to be used for operating absolute encoders on the EnDat interface. The DioCl2EnDatIn object must already have been created by using [DioCl2EnDatIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EnDatIn_create.....	538
DioCl2EnDatIn_read.....	551
DioCl2EnDatIn_start.....	565

Hall Sensor Interface

Introduction

You can measure the position of a motor's rotor by using an angle computation unit and the signals of a Hall sensor that is connected to DIO Class 1 or DIO Class 2 channels.

Where to go from here

Information in this section

Hall Sensor Class 1	568
Hall Sensor Class 2	586

Hall Sensor Class 1

Introduction

Hall sensors that are connected to DIO Class 1 channels can be accessed via the [DioCl1HallIn](#) functions.

Where to go from here

Information in this section

DioCl1HallIn_apply	569
To apply the initialization settings to the DIO Class 1 Hall input channels.	
DioCl1HallIn_create	570
To create the I/O driver object for Hall sensor signal measurement via DIO Class 1 input channels.	
DioCl1HallIn_getIsPositionValid	573
To verify whether the provided electrical sensor position is valid.	
DioCl1HallIn_getPosition	574
To get the current electrical sensor position.	
DioCl1HallIn_getSensorSignals	576
To get the current states of the Hall sensor signals captured on DIO class 1 input channels.	
DioCl1HallIn_read	577
To read data from a Hall sensor via DIO Class 1 input channels.	
DioCl1HallIn_setInputFilter	578
To specify the time interval for the noise filtering of Hall sensor signals.	

DioCl1HallIn_setPositionOffset	580
To specify an offset angle for the evaluated rotor position.	
DioCl1HallIn_setReverseDirection	581
To specify that the Hall sensor signals are interpreted as a reverse rotation.	
DioCl1HallIn_setSigEdgePositions	582
To specify the positions of the single signal edges of the Hall sensor.	
DioCl1HallIn_start	583
To start measuring Hall sensor signals via DIO Class 1 input channels.	
DioCl1HallIn_write	584
To update the settings of Hall sensor signal measurement via DIO Class 1 input channels during run time.	

DioCl1HallIn_apply

Syntax

```
UInt32 DioCl1HallIn_apply(
    DioCl1HallInSDrvObject *pDioCl1HallIn)
```

Include file

`IoDrvDicClass1HallIn.h`

Purpose

To apply the initialization settings of Hall sensor signal measurement via DIO Class 1 input channels.

Description

Before you can start to measure the Hall sensor signals via the specified DIO Class 1 channels, you must transfer the settings of the DioCl1HallIn driver object to its related hardware by using `DioCl1HallIn_apply`.

You must do this also for the default values and for values that you specified via the `DioCl1HallIn_set<Parameter>` functions. The `DioCl1HallIn_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The signal measurement on the specified digital input channels is not activated until you call `DioCl1HallIn_start`.

To update settings during run time, you must use `DioCl1HallIn_write`.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [Hall Sensor Interface](#) ([MicroLabBox Features](#) ).

Parameters **pDioCl1HallIn** Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using [DioCl1HallIn_create](#).

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1HallIn_create.....	570
DioCl1HallIn_SetInputFilter.....	578
DioCl1HallIn_SetPositionOffset.....	580
DioCl1HallIn_SetReverseDirection.....	581
DioCl1HallIn_SetSigEdgePositions.....	582
DioCl1HallIn_Start.....	583

DioCl1HallIn_create

Syntax

```
UInt32 DioCl1HallIn_create(
    DioCl1HallInSDrvObject **ppDioCl1HallIn,
    UInt32 HallInUnitNo,
    UInt32 Port,
    UInt32 Channel,
    UInt32 HallSensorCount)
```

Include file

`IoDrvDioClass1HallIn.h`

Purpose

To create the I/O driver object for Hall sensor signal measurement via DIO Class 1 input channels.

Description This function is used to perform all the steps necessary to create an I/O driver object to access DIO Class 1 channels.

A DioCl1HallIn driver object handles one Hall sensor interface with input channels for *Signal A*, *Signal B* and *Signal C*.

You allocate the required consecutive channels by specifying the first channel and the number of channels (currently always three channels).

The Hall sensor I/O driver is initialized with default values.

The initial values are:

- Rotational direction: Not inverted (according to the order of the Hall sensor signals)
- Input filter: 0.0 s
- Offset added to resulting angle position: 0°
- Positions of Hall sensors defined by rising and falling edges of sensor signals:

Sensor Input	Rising Edge	Falling Edge
Hall A	0°	180°
Hall B	120°	300°
Hall C	240°	60°

The function is intended to be called during the initialization phase of the real-time application.

You can use the `DioCl1HallIn_set<Parameter>` functions to change the default parameters. Before you can start the Hall sensor interface functionality, you have to transfer the settings of the DioCl1HallIn driver object to its related hardware by using `DioCl1HallIn_apply`.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

ppDioCl1HallIn Lets you specify the address of a variable which holds the address of the created driver object.

HallInUnitNo Lets you specify an instance number for the created driver object of the Hall sensor in the range 1 ... 2. One driver object can handle one Hall sensor interface. You can use the instance number to attach the sensor to an angle computation unit via `Acu_setInputSensor`.

Symbol	Meaning
DIO_HALL_IN_UNIT_1	Specifies Hall sensor interface 1.
DIO_HALL_IN_UNIT_2	Specifies Hall sensor interface 2.

Port Lets you specify the number of the DIO Class 1 port that provides the required channels in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
...	...
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

Channel Lets you specify the channel to be used for *Signal A* of the connected Hall sensor in the range 1 ... 14. The following two channels are used for *Signal B* and *Signal C*. The channels used by the Hall sensor must not be allocated by other DIO Class 1 functions.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified DIO Class 1 port.
...	...
DIO_CLASS1_CHANNEL_14	Specifies channel 14 of the specified DIO Class 1 port.

HallSensorCount Lets you specify the number of sensor signals to be handled by the Hall sensor interface. You can specify three sensor signals by using DIO_HALL_SENSOR_COUNT_3.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1HallIn_apply.....	569
DioCl1HallIn_setInputFilter.....	578
DioCl1HallIn_setPositionOffset.....	580
DioCl1HallIn_setReverseDirection.....	581
DioCl1HallIn_setSigEdgePositions.....	582
DioCl1HallIn_start.....	583

DioCl1HallIn_getIsPositionValid

Syntax

```
UInt32 DioCl1HallIn_getIsPositionValid(
    DioCl1HallInSDrvObject *pDioCl1HallIn,
    UInt32 *pIsPositionValid)
```

Include file

`IoDrvDioClass1HallIn.h`

Purpose

To verify whether the electrical sensor position is valid.

Description

This function is used to evaluate the electrical sensor position that is provided by [DioCl1HallIn_getPosition](#).

An invalid value occurs if the Hall sensor provides a signal pattern that cannot be interpreted as a unique rotor position, for example, if all signals are *Low* as a result of a cable break. For more information on the signal patterns of Hall sensors, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call [DioCl1HallIn_read](#) to update the internal buffer with the latest values.

Note

All digital inputs of the DIO Class 1 unit are in a disabled state after reset and will read 0. To enable the Hall sensor input channels that you specified by using [DioCl1HallIn_create](#), you must use [DioCl1HallIn_start](#).

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1HallIn Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using [DioCl1HallIn_create](#).

pisPositionValid Lets you specify the address of a variable that holds the result of the validity check.

Value	Meaning
0	The mechanical rotor position value is invalid.
> 0	The mechanical rotor position value is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1HallIn_getPosition.....	574
DioCl1HallIn_getSensorSignals	576
DioCl1HallIn_read.....	577
DioCl1HallIn_start.....	583

DioCl1HallIn_getPosition

Syntax

```
UInt32 DioCl1HallIn_getPosition(
    DioCl1HallInSDrvObject *pDioCl1HallIn,
    Float64 *pPosition)
```

Include file

IoDrvDioClass1HallIn.h

Purpose

To get the current electrical sensor position.

Description

This function is used to provide the current electrical sensor position that is evaluated based on the following read sensor data:

- The digital state of the three Hall sensor input channels
- The geometry of the falling and rising edges of the three Hall sensor input channels that you can adjust via **DioCl1HallIn_setSigEdgePositions**
- The position offset that is added to the calculated electrical position that you can specify via **DioCl1HallIn_setPositionOffset**

The Hall sensor detects its position by sector. The electrical position provided by **DioCl1HallIn_getPosition** is the mean of the two angles that limit the particular sector. For example, the provided electrical sensor position for the sector between 120° and 180° is 150°.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl1HallIn_read** to update the internal buffer with the latest values.

Note

All digital inputs of the DIO Class 1 unit are in a disabled state after reset and will read 0. To enable the Hall sensor input channels that you specified by using **DioCl1HallIn_create**, you must use **DioCl1HallIn_start**.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1HallIn Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using **DioCl1HallIn_create**.

pPosition Lets you specify the address of a variable that holds the current electrical sensor position in degrees in the range 0° ... 359.9945° with a resolution of 0.0055°.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1HallIn_getIsPositionValid.....	573
DioCl1HallIn_getSensorSignals	576
DioCl1HallIn_read.....	577
DioCl1HallIn_start.....	583

DioCl1HallIn_getSensorSignals

Syntax	<pre>UInt32 DioCl1HallIn_getSensorSignals(DioCl1HallInSDrvObject *pDioCl1HallIn, UInt32 *pSensorSignalStates)</pre>
Include file	<code>IoDrvDioClass1HallIn.h</code>
Purpose	To get the current states of the Hall sensor signals captured on DIO class 1 input channels.
Description	<p>This function is used to provide the digital states (0 or 1) of the three Hall sensor input channels.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call <code>DioCl1HallIn_read</code> to update the internal buffer with the latest values.</p>
	<p>Note</p> <p>All digital inputs of the DIO Class 1 unit are in a disabled state after reset and will read 0. To enable the Hall sensor input channels that you specified by using <code>DioCl1HallIn_create</code>, you must use <code>DioCl1HallIn_start</code>.</p>
	If an error is detected, the first instance of the error is returned to the caller.
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Hall Sensor Interface (MicroLabBox Features).</p>
Parameters	<p>pDioCl1HallIn Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using <code>DioCl1HallIn_create</code>.</p> <p>pSensorSignalStates Lets you specify the address of an array that holds the current digital states (0 or 1) of the Hall sensor signals.</p> <p>The first value specifies the state of signal A, the second of signal B and the third of signal C.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1HallIn_getIsPositionValid.....	573
DioCl1HallIn_getPosition.....	574
DioCl1HallIn_read.....	577
DioCl1HallIn_start.....	583

DioCl1HallIn_read

Syntax

```
UInt32 DioCl1HallIn_read(DioCl1HallInSDrvObject *pDioCl1HallIn)
```

Include file

IoDrvDioClass1HallIn.h

Purpose

To read data from a Hall sensor via DIO Class 1 input channels.

Description

This function is used to read the raw data of the DIO Class 1 channels that are connected to the Hall sensor. It checks whether the captured signals are valid. Then it evaluates the rotor position and stores the results as a consistent data set in an internal buffer.

The input channels must be enabled before via **DioCl1HallIn_start**.

To get the values from the internal buffer, you must use one of the following functions:

- **DioCl1HallIn_getPosition** to get the value of the rotor position
- **DioCl1HallIn_getIsPositionValid** to get the information on whether the value of the electrical rotor position is valid
- **DioCl1HallIn_getSensorSignals** to get the current states of the Hall sensor signals

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration ).
	For more information on the I/O mapping, refer to Hall Sensor Interface (MicroLabBox Features ).
Parameters	pDioCl1HallIn Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using DioCl1HallIn_create .
Return value	The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1HallIn_getIsPositionValid.....	573
DioCl1HallIn_getPosition.....	574
DioCl1HallIn_getSensorSignals	576
DioCl1HallIn_start.....	583

DioCl1HallIn_setInputFilter

Syntax	<pre>UInt32 DioCl1HallIn_setInputFilter(DioCl1HallInSDrvObject *pDioCl1HallIn, Float64 FilterInterval)</pre>
Include file	<code>IoDrvDioClass1HallIn.h</code>
Purpose	To specify the time interval for the noise filtering of Hall sensor signals on DIO Class 1 input channels.
Description	<p>Only those input signals will be processed which have constant values over a period of time longer than the <code>FilterInterval</code> parameter specifies.</p> <p>The input filter will be applied to all the channels that are controlled by the specified DioCl1HallIn driver object.</p>

The setting does not take effect until you call **DioCl1HallIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1HallIn Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using **DioCl1HallIn_create**.

FilterInterval Lets you specify the minimum time interval for which the input signal(s) must provide a constant value that is to be used for processing. Shorter input signals will be filtered. You can specify filter intervals in seconds in the range 0 ... 0.01 s.

The resolution/deviation depends on the specified range:

- 0 ... 640 ns
Resolution of 10 ns
- >640 ns ... 0.01 s
Deviation of <10%

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1HallIn_apply.....	569
DioCl1HallIn_create.....	570

DioCl1HallIn_setPositionOffset

Syntax

```
UInt32 DioCl1HallIn_setPositionOffset(
    DioCl1HallInSDrvObject *pDioCl1HallIn,
    Float64 PositionOffset)
```

Include file

`IoDrvDioClass1HallIn.h`

Purpose

To specify an offset angle for the evaluated rotor position.

Description

This function is used to specify an offset angle for the measured angular positions of the Hall sensor signals.

The setting does not take effect until you call `DioCl1HallIn_apply` during the initialization phase or `DioCl1HallIn_write` during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1HallIn Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using `DioCl1HallIn_create`.

PositionOffset Lets you specify the position offset in degrees in the range -60° ... +60° with a resolution of 0.0055°..

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics

References

DioCl1HallIn_apply.....	569
DioCl1HallIn_start.....	583
DioCl1HallIn_write.....	584

DioCl1HallIn_setReverseDirection

Syntax

```
UInt32 DioCl1HallIn_setReverseDirection(
    DioCl1HallInSDrvObject *pDioCl1HallIn,
    UInt32 Reversing)
```

Include file

IoDrvDioClass1HallIn.h

Purpose

To specify that the Hall sensor signals are interpreted as a reverse rotation.

Description

This function is useful if interchanged signals in a cable harness are to be compensated.

The setting does not take effect until you call **DioCl1HallIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1HallIn Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using **DioCl1HallIn_create**.

Reversing Lets you specify whether the Hall sensor signals are interpreted as a reverse rotation.

Symbol	Meaning
DIO_HALL_REVERSE_INACTIVE	Specifies to interpret the Hall sensor signals as a straight rotation according to the order of the Hall sensor signals (default).

Symbol	Meaning
DIO_HALL_REVERSE_ACTIVE	Specifies to interpret the Hall sensor signals as a reverse rotation according to the order of the Hall sensor signals.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1HallIn_apply.....	569
DioCl1HallIn_create.....	570

DioCl1HallIn_setSigEdgePositions

Syntax

```
UInt32 DioCl1HallIn_setSigEdgePositions(
    DioCl1HallInSDrvObject *pDioCl1HallIn,
    Float64 *pSignalEdgePositions)
```

Include file

IoDrvDioClass1HallIn.h

Purpose

To specify the positions of the signal edges of the Hall sensor.

Description

This function is used to specify the angular position of the rising and the falling edges for each Hall sensor signal separately.

The setting does not take effect until you call **DioCl1HallIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1HallIn Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using [DioCl1HallIn_create](#).

pSignalEdgePositions Lets you specify the address of an array that stores the angular positions of the rising and falling edges in degrees.

The array must contain three pairs of angles. The first pair specifies signal A, the second signal B and the third signal C. Within each pair, the first angle specifies the position of the rising edge and the second the position of the falling edge. For example, the array {0, 180, 120, 300, 240, 60} specifies the default values.

The range of valid angular positions is 0.0° ... 360.0° with a resolution of 0.0055°.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1HallIn_apply.....	569
DioCl1HallIn_create.....	570

[DioCl1HallIn_start](#)**Syntax**

```
UInt32 DioCl1HallIn_start(
    DioCl1HallInSDrvObject *pDioCl1HallIn)
```

Include file

`IoDrvDioClass1HallIn.h`

Purpose

To start measuring Hall sensor signals via DIO Class 1 input channels.

Description	This function is used to start the specified DioCl1HallIn driver so that it will activate its digital input channels. The digital input channels are enabled to start capturing the Hall sensor input signals. The function is intended to be called during the run time of the real-time application. If an error is detected, the first instance of the error is returned to the caller.
I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) . For more information on the I/O mapping, refer to Hall Sensor Interface (MicroLabBox Features) .
Parameters	pDioCl1HallIn Lets you specify the target Hall sensor via the corresponding DioCl1HallIn driver object. The DioCl1HallIn driver object had to be created before by using DioCl1HallIn_create .
Return value	The function returns an error code.

References	
DioCl1HallIn_apply.....	569
DioCl1HallIn_create.....	570
DioCl1HallIn_read.....	577
DioCl1HallIn_write.....	584

DioCl1HallIn_write

Syntax	<pre>UInt32 DioCl1HallIn_write(DioCl1HallInSDrvObject *pDioCl1HallIn)</pre>
Include file	<code>IoDrvDioClass1HallIn.h</code>

Purpose To update the settings of Hall sensor signal measurement via DIO Class 1 input channels during run time.

Description If you have used the following function during run time, its settings will not be updated on the hardware until you call `DioCl1HallIn_write`:

- **`DioCl1HallIn_setPositionOffset`**

The specified digital input channels must be enabled before by using `DioCl1HallIn_start`.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters **`pDioCl1HallIn`** Lets you specify the target Hall sensor via the corresponding `DioCl1HallIn` driver object. The `DioCl1HallIn` driver object had to be created before by using `DioCl1HallIn_create`.

Return value The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics

References

<code>DioCl1HallIn_apply</code>	569
<code>DioCl1HallIn_create</code>	570
<code>DioCl1HallIn_setPositionOffset</code>	580
<code>DioCl1HallIn_start</code>	583

Hall Sensor Class 2

Introduction	Hall sensors that are connected to DIO Class 2 channels can be accessed via the DioCl2HallIn functions.
Where to go from here	Information in this section
	<p>DioCl2HallIn_apply.....587 To apply the initialization settings to the DIO Class 2 Hall input channels.</p> <p>DioCl2HallIn_create.....588 To create the I/O driver object for Hall sensor signal measurement via DIO Class 2 input channels.</p> <p>DioCl2HallIn_getIsPositionValid.....590 To verify whether the provided electrical sensor position is valid.</p> <p>DioCl2HallIn_getPosition.....592 To get the current electrical sensor position.</p> <p>DioCl2HallIn_getSensorSignals</p> <p>To get the current states of the Hall sensor signals captured on DIO class 2 input channels.</p> <p>DioCl2HallIn_read.....594 To read data from a Hall sensor via DIO Class 2 input channels.</p> <p>DioCl2HallIn_setInputFilter.....596 To specify the time interval for the noise filtering of Hall sensor signals.</p> <p>DioCl2HallIn_setPositionOffset.....597 To specify an offset angle for the evaluated rotor position.</p> <p>DioCl2HallIn_setReverseDirection.....598 To specify that the Hall sensor signals are interpreted as a reverse rotation.</p> <p>DioCl2HallIn_setSigEdgePositions.....599 To specify the positions of the single signal edges of the Hall sensor.</p> <p>DioCl2HallIn_start.....601 To start measuring Hall sensor signals via DIO Class 2 input channels.</p> <p>DioCl2HallIn_write.....602 To update the settings of Hall sensor signal measurement via DIO Class 2 input channels during run time.</p>

DioCl2HallIn_apply

Syntax

```
UInt32 DioCl2HallIn_apply(
    DioCl2HallInSDrvObject *pDioCl2HallIn)
```

Include file

`IoDrvDioClass2HallIn.h`

Purpose

To apply the initialization settings of Hall sensor signal measurement via DIO Class 2 input channels.

Description

Before you can start to measure the Hall sensor signals via the specified DIO Class 2 channels, you must transfer the settings of the `DioCl2HallIn` driver object to its related hardware by using `DioCl2HallIn_apply`.

You must do this also for the default values and for values that you specified via the `DioCl2HallIn_set<Parameter>` functions. The `DioCl2HallIn_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The signal measurement on the specified digital input channels is not activated until you call `DioCl2HallIn_start`.

To update settings during run time, you must use `DioCl2HallIn_write`.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

`pDioCl2HallIn` Lets you specify the target Hall sensor via the corresponding `DioCl2HallIn` driver object. The `DioCl2HallIn` driver object had to be created before by using `DioCl2HallIn_create`.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics**References**

DioCl2HallIn_create.....	588
DioCl2HallIn_setInputFilter.....	596
DioCl2HallIn_setPositionOffset.....	597
DioCl2HallIn_setReverseDirection.....	598
DioCl2HallIn_setSigEdgePositions.....	599
DioCl2HallIn_start.....	601

DioCl2HallIn_create

Syntax

```
UInt32 DioCl2HallIn_create(
    DioCl2HallInSDrvObject **ppDioCl2HallIn,
    UInt32 HallInUnitNo,
    UInt32 Channel,
    UInt32 HallSensorCount)
```

Include file

IoDrvDioClass2HallIn.h

Purpose

To create the I/O driver object for Hall sensor signal measurement via DIO Class 2 input channels.

Description

This function is used to perform all the steps necessary to create an I/O driver object to access DIO Class 2 channels.

A DioCl2HallIn driver object handles one Hall sensor interface with input channels for *Signal A*, *Signal B* and *Signal C*.

You allocate the required consecutive channels by specifying the first channel and the number of channels (currently always three channels).

The Hall sensor I/O driver is initialized with default values.

The initial values are:

- Rotational direction: Not inverted (according to the order of the Hall sensor signals)
- Input filter: 0.0 s
- Offset added to resulting angle position: 0°

- Positions of Hall sensors defined by rising and falling edges of sensor signals:

Sensor Input	Rising Edge	Falling Edge
Hall A	0°	180°
Hall B	120°	300°
Hall C	240°	60°

The function is intended to be called during the initialization phase of the real-time application.

You can use the `DioCl2HallIn_set<Parameter>` functions to change the default parameters. Before you can start the Hall sensor interface functionality, you have to transfer the settings of the DioCl2HallIn driver object to its related hardware by using `DioCl2HallIn_apply`.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

ppDioCl2HallIn Lets you specify the address of a variable which holds the address of the created driver object.

HallInUnitNo Lets you specify an instance number for the created driver object of the Hall sensor in the range 1 ... 2. One driver object can handle one Hall sensor interface. You can use the instance number to attach the sensor to an angle computation unit via `Acu_setInputSensor`.

Symbol	Meaning
DIO_HALL_IN_UNIT_1	Specifies Hall sensor interface 1.
DIO_HALL_IN_UNIT_2	Specifies Hall sensor interface 2.

Channel Lets you specify the channel to be used for *Signal A* of the connected Hall sensor in the range 1 ... 10. The following two channels are used for *Signal B* and *Signal C*. The channels used by the Hall sensor must not be allocated by other DIO Class 2 functions.

Symbol	Meaning
DIO_CLASS2_CHANNEL_1	Specifies channel 1
...	...
DIO_CLASS2_CHANNEL_10	Specifies channel 10.

HallSensorCount Lets you specify the number of sensor signals to be handled by the Hall sensor interface. You can specify three sensor signals by using `DIO_HALL_SENSOR_COUNT_3`.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2HallIn_apply.....	587
DioCl2HallIn_setInputFilter.....	596
DioCl2HallIn_setPositionOffset.....	597
DioCl2HallIn_setReverseDirection.....	598
DioCl2HallIn_setSigEdgePositions.....	599
DioCl2HallIn_start.....	601

DioCl2HallIn_getIsPositionValid

Syntax

```
UInt32 DioCl2HallIn_getIsPositionValid(
    DioCl2HallInSDrvObject *pDioCl2HallIn,
    UInt32 *pIsPositionValid)
```

Include file

IoDrvDioClass2HallIn.h

Purpose

To verify whether the electrical sensor position is valid.

Description

This function is used to evaluate the electrical sensor position that is provided by [DioCl2HallIn_getPosition](#).

An invalid value occurs if the Hall sensor provides a signal pattern that cannot be interpreted as a unique rotor position, for example, if all signals are *Low* as a result of a cable break. For more information on the signal patterns of Hall sensors, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call [DioCl2HallIn_read](#) to update the internal buffer with the latest values.

Note

All digital inputs of the DIO Class 2 unit are in a disabled state after reset and will read 0. To enable the Hall sensor input channels that you specified by using `DioCl2HallIn_create`, you must use `DioCl2HallIn_start`.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding DioCl2HallIn driver object. The DioCl2HallIn driver object had to be created before by using `DioCl2HallIn_create`.

plsPositionValid Lets you specify the address of a variable that holds the result of the validity check.

Value	Meaning
0	The mechanical rotor position value is invalid.
> 0	The mechanical rotor position value is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2HallIn_getPosition.....	592
DioCl2HallIn_getSensorSignals	593
DioCl2HallIn_read.....	594
DioCl2HallIn_start.....	601

DioCl2HallIn_getPosition

Syntax	<pre>UInt32 DioCl2HallIn_getPosition(DioCl2HallInSDrvObject *pDioCl2HallIn, Float64 *pPosition)</pre>
Include file	IoDrvDioClass2HallIn.h
Purpose	To get the current electrical sensor position.
Description	<p>This function is used to provide the current electrical sensor position that is evaluated based on the following read sensor data:</p> <ul style="list-style-type: none"> ▪ The digital state of the three Hall sensor input channels ▪ The geometry of the falling and rising edges of the three Hall sensor input channels that you can adjust via DioCl2HallIn_setSigEdgePositions ▪ The position offset that is added to the calculated electrical position that you can specify via DioCl2HallIn_setPositionOffset <p>The Hall sensor detects its position by sector. The electrical position provided by DioCl2HallIn_getPosition is the mean of the two angles that limit the particular sector. For example, the provided electrical sensor position for the sector between 120° and 180° is 150°.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call DioCl2HallIn_read to update the internal buffer with the latest values.</p>
Note	<p>All digital inputs of the DIO Class 2 unit are in a disabled state after reset and will read 0. To enable the Hall sensor input channels that you specified by using DioCl2HallIn_create, you must use DioCl2HallIn_start.</p>
I/O mapping	<p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>
	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Hall Sensor Interface (MicroLabBox Features).</p>

Parameters

pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding DioCl2HallIn driver object. The DioCl2HallIn driver object had to be created before by using **DioCl2HallIn_create**.

pPosition Lets you specify the address of a variable that holds the current electrical sensor position in degrees in the range 0° ... 359.9945° with a resolution of 0.0055°.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2HallIn_getIsPositionValid.....	590
DioCl2HallIn_getSensorSignals	593
DioCl2HallIn_read.....	594
DioCl2HallIn_start.....	601

DioCl2HallIn_getSensorSignals

Syntax

```
UInt32 DioCl2HallIn_getSensorSignals(
    DioCl2HallInSDrvObject *pDioCl2HallIn,
    UInt32 *pSensorSignalStates)
```

Include file

IoDrvDioClass2HallIn.h

Purpose

To get the current states of the Hall sensor signals captured on DIO class 2 input channels.

Description

This function is used to provide the digital states (0 or 1) of the three Hall sensor input channels.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl2HallIn_read** to update the internal buffer with the latest values.

Note

All digital inputs of the DIO Class 2 unit are in a disabled state after reset and will read 0. To enable the Hall sensor input channels that you specified by using `DioCl2HallIn_create`, you must use `DioCl2HallIn_start`.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding DioCl2HallIn driver object. The DioCl2HallIn driver object had to be created before by using `DioCl2HallIn_create`.

pSensorSignalStates Lets you specify the address of an array that holds the current digital states (0 or 1) of the Hall sensor signals.

The first value specifies the state of signal A, the second of signal B and the third of signal C.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2HallIn_getIsPositionValid.....	590
DioCl2HallIn_getPosition.....	592
DioCl2HallIn_read.....	594
DioCl2HallIn_start.....	601

DioCl2HallIn_read

Syntax

```
UInt32 DioCl2HallIn_read(DioCl2HallInSDrvObject *pDioCl2HallIn)
```

Include file	IoDrvDioClass2HallIn.h						
Purpose	To read data from a Hall sensor via DIO Class 2 input channels.						
Description	<p>This function is used to read the raw data of the DIO Class 2 channels that are connected to the Hall sensor. It checks whether the captured signals are valid. Then it evaluates the rotor position and stores the results as a consistent data set in an internal buffer.</p> <p>The input channels must be enabled before via DioCl2HallIn_start.</p> <p>To get the values from the internal buffer, you must use one of the following functions:</p> <ul style="list-style-type: none"> ▪ DioCl2HallIn_getPosition to get the value of the rotor position ▪ DioCl2HallIn_getIsPositionValid to get the information on whether the value of the electrical rotor position is valid ▪ DioCl2HallIn_getSensorSignals to get the current states of the Hall sensor signals <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Hall Sensor Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding DioCl2HallIn driver object. The DioCl2HallIn driver object had to be created before by using DioCl2HallIn_create.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References
	<p>DioCl2HallIn_getIsPositionValid..... 590 DioCl2HallIn_getPosition..... 592 DioCl2HallIn_getSensorSignals 593 DioCl2HallIn_start..... 601</p>

DioCl2HallIn_setInputFilter

Syntax	<pre>UInt32 DioCl2HallIn_setInputFilter(DioCl2HallInSDrvObject *pDioCl2HallIn, Float64 FilterInterval)</pre>
Include file	IoDrvDioClass2HallIn.h
Purpose	To specify the time interval for the noise filtering of Hall sensor signals on DIO Class 2 input channels.
Description	<p>Only those input signals will be processed which have constant values over a period of time longer than the FilterInterval parameter specifies.</p> <p>The input filter will be applied to all the channels that are controlled by the specified DioCl2HallIn driver object.</p> <p>The setting does not take effect until you call DioCl2HallIn_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Hall Sensor Interface (MicroLabBox Features).</p>
Parameters	<p>pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding DioCl2HallIn driver object. The DioCl2HallIn driver object had to be created before by using DioCl2HallIn_create.</p> <p>FilterInterval Lets you specify the minimum time interval for which the input signal(s) must provide a constant value that is to be used for processing. Shorter</p>

input signals will be filtered. You can specify filter intervals in seconds in the range 0 ... 0.01 s.

The resolution/deviation depends on the specified range:

- 0 ... 640 ns
Resolution of 10 ns
- >640 ns ... 0.01 s
Deviation of <10%

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2HallIn_apply.....	587
DioCl2HallIn_create.....	588

DioCl2HallIn_setPositionOffset

Syntax

```
UInt32 DioCl2HallIn_setPositionOffset(
    DioCl2HallInSDrvObject *pDioCl2HallIn,
    Float64 PositionOffset)
```

Include file

IoDrvDioClass2HallIn.h

Purpose

To specify an offset angle for the evaluated rotor position.

Description

This function is used to specify an offset angle for the measured angular positions of the Hall sensor signals.

The setting does not take effect until you call **DioCl2HallIn_apply** during the initialization phase or **DioCl2HallIn_write** during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration ).
	For more information on the I/O mapping, refer to Hall Sensor Interface (MicroLabBox Features ).
Parameters	<p>pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding DioCl2HallIn driver object. The DioCl2HallIn driver object had to be created before by using DioCl2HallIn_create.</p> <p>PositionOffset Lets you specify the position offset in degrees in the range -60° ... +60° with a resolution of 0.0055°..</p>
Return value	The function returns an error code.

Related topics	References
	DioCl2HallIn_apply 587
	DioCl2HallIn_start 601
	DioCl2HallIn_write 602

DioCl2HallIn_setReverseDirection

Syntax	<pre>UInt32 DioCl2HallIn_setReverseDirection(DioCl2HallInSDrvObject *pDioCl2HallIn, UInt32 Reversing)</pre>
Include file	<code>IoDrvDioClass2HallIn.h</code>
Purpose	To specify that the Hall sensor signals are interpreted as a reverse rotation.
Description	This function is useful if interchanged signals in a cable harness are to be compensated.

The setting does not take effect until you call **DioCl2HallIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding DioCl2HallIn driver object. The DioCl2HallIn driver object had to be created before by using **DioCl2HallIn_create**.

Reversing Lets you specify whether the Hall sensor signals are interpreted as a reverse rotation.

Symbol	Meaning
DIO_HALL_REVERSE_INACTIVE	Specifies to interpret the Hall sensor signals as a straight rotation according to the order of the Hall sensor signals (default).
DIO_HALL_REVERSE_ACTIVE	Specifies to interpret the Hall sensor signals as a reverse rotation according to the order of the Hall sensor signals.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2HallIn_apply.....	587
DioCl2HallIn_create.....	588

DioCl2HallIn_setSigEdgePositions

Syntax

```
UInt32 DioCl2HallIn_setSigEdgePositions(
    DioCl2HallInSDrvObject *pDioCl2HallIn,
    Float64 *pSignalEdgePositions)
```

Include file	<code>IoDrvDioClass2HallIn.h</code>						
Purpose	To specify the positions of the signal edges of the Hall sensor.						
Description	<p>This function is used to specify the angular position of the rising and the falling edges for each Hall sensor signal separately.</p> <p>The setting does not take effect until you call <code>DioCl2HallIn_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Hall Sensor Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding DioCl2HallIn driver object. The DioCl2HallIn driver object had to be created before by using <code>DioCl2HallIn_create</code>.</p> <p>pSignalEdgePositions Lets you specify the address of an array that stores the angular positions of the rising and falling edges in degrees.</p> <p>The array must contain three pairs of angles. The first pair specifies signal A, the second signal B and the third signal C. Within each pair, the first angle specifies the position of the rising edge and the second the position of the falling edge. For example, the array {0, 180, 120, 300, 240, 60} specifies the default values.</p> <p>The range of valid angular positions is 0.0° ... 360.0° with a resolution of 0.0055°.</p>						
Return value	The function returns an error code.						
Related topics	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td> <td>The function was successfully completed.</td> </tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						
References	<p><code>DioCl2HallIn_apply</code>..... 587 <code>DioCl2HallIn_create</code>..... 588</p>						

DioCl2HallIn_start

Syntax

```
UInt32 DioCl2HallIn_start(
    DioCl2HallInSDrvObject *pDioCl2HallIn)
```

Include file

`IoDrvDioClass2HallIn.h`

Purpose

To start measuring Hall sensor signals via DIO Class 2 input channels.

Description

This function is used to start the specified DioCl2HallIn driver so that it will activate its digital input channels. The digital input channels are enabled to start capturing the Hall sensor input signals.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding DioCl2HallIn driver object. The DioCl2HallIn driver object had to be created before by using [DioCl2HallIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2HallIn_apply.....	587
DioCl2HallIn_create.....	588
DioCl2HallIn_read.....	594
DioCl2HallIn_write.....	602

DioCl2HallIn_write

Syntax

```
UInt32 DioCl2HallIn_write(
    DioCl2HallInSDrvObject *pDioCl2HallIn)
```

Include file

`IoDrvDioClass2HallIn.h`

Purpose

To update the settings of Hall sensor signal measurement via DIO Class 2 input channels during run time.

Description

If you have used the following function during run time, its settings will not be updated on the hardware until you call `DioCl2HallIn_write`:

- `DioCl2HallIn_setPositionOffset`

The specified digital input channels must be enabled before by using `DioCl2HallIn_start`.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Hall Sensor Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2HallIn Lets you specify the target Hall sensor via the corresponding `DioCl2HallIn` driver object. The `DioCl2HallIn` driver object had to be created before by using `DioCl2HallIn_create`.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics**References**

DioCl2HallIn_apply.....	587
DioCl2HallIn_create.....	588
DioCl2HallIn_setPositionOffset.....	597
DioCl2HallIn_start.....	601

Incremental Encoder Interface

Introduction	You can measure the rotor position of an electric motor by using the signals of an incremental encoder that is connected to DIO Class 1 or DIO Class 2 channels.
---------------------	--

Where to go from here	Information in this section				
	<table> <tr> <td> Incremental Encoder Class 1.....</td> <td>604</td> </tr> <tr> <td> Incremental Encoder Class 2.....</td> <td>635</td> </tr> </table>	Incremental Encoder Class 1.....	604	Incremental Encoder Class 2.....	635
Incremental Encoder Class 1.....	604				
Incremental Encoder Class 2.....	635				

Incremental Encoder Class 1

Introduction	Incremental encoders that are connected to DIO Class 1 channels can be accessed via the <code>DioCl1EncoderIn</code> functions.
---------------------	---

Where to go from here	Information in this section																																
	<table> <tr> <td> DioCl1EncoderIn_apply.....</td> <td>606</td> </tr> <tr> <td> To apply the initialization settings to the DIO Class 1 incremental encoder channels.</td> <td></td> </tr> <tr> <td> DioCl1EncoderIn_create.....</td> <td>607</td> </tr> <tr> <td> To create the I/O driver object for incremental encoder signal measurement via DIO Class 1 input channels.</td> <td></td> </tr> <tr> <td> DioCl1EncoderIn_getEncPosition.....</td> <td>609</td> </tr> <tr> <td> To get the current encoder position in lines from the position counter.</td> <td></td> </tr> <tr> <td> DioCl1EncoderIn_getEncVelocity.....</td> <td>611</td> </tr> <tr> <td> To get the encoder velocity in lines per second.</td> <td></td> </tr> <tr> <td> DioCl1EncoderIn_getIsIndexRaised.....</td> <td>612</td> </tr> <tr> <td> To verify whether a first index signal pulse occurred.</td> <td></td> </tr> <tr> <td> DioCl1EncoderIn_getMechPosition.....</td> <td>613</td> </tr> <tr> <td> To get the mechanical position of the encoder in degrees.</td> <td></td> </tr> <tr> <td> DioCl1EncoderIn_getMechVelocity.....</td> <td>614</td> </tr> <tr> <td> To get the mechanical velocity of the encoder in revolutions per minute.</td> <td></td> </tr> <tr> <td> DioCl1EncoderIn_read.....</td> <td>616</td> </tr> <tr> <td> To read data from an incremental encoder via DIO Class 1 input channels.</td> <td></td> </tr> </table>	DioCl1EncoderIn_apply	606	To apply the initialization settings to the DIO Class 1 incremental encoder channels.		DioCl1EncoderIn_create	607	To create the I/O driver object for incremental encoder signal measurement via DIO Class 1 input channels.		DioCl1EncoderIn_getEncPosition	609	To get the current encoder position in lines from the position counter.		DioCl1EncoderIn_getEncVelocity	611	To get the encoder velocity in lines per second.		DioCl1EncoderIn_getIsIndexRaised	612	To verify whether a first index signal pulse occurred.		DioCl1EncoderIn_getMechPosition	613	To get the mechanical position of the encoder in degrees.		DioCl1EncoderIn_getMechVelocity	614	To get the mechanical velocity of the encoder in revolutions per minute.		DioCl1EncoderIn_read	616	To read data from an incremental encoder via DIO Class 1 input channels.	
DioCl1EncoderIn_apply	606																																
To apply the initialization settings to the DIO Class 1 incremental encoder channels.																																	
DioCl1EncoderIn_create	607																																
To create the I/O driver object for incremental encoder signal measurement via DIO Class 1 input channels.																																	
DioCl1EncoderIn_getEncPosition	609																																
To get the current encoder position in lines from the position counter.																																	
DioCl1EncoderIn_getEncVelocity	611																																
To get the encoder velocity in lines per second.																																	
DioCl1EncoderIn_getIsIndexRaised	612																																
To verify whether a first index signal pulse occurred.																																	
DioCl1EncoderIn_getMechPosition	613																																
To get the mechanical position of the encoder in degrees.																																	
DioCl1EncoderIn_getMechVelocity	614																																
To get the mechanical velocity of the encoder in revolutions per minute.																																	
DioCl1EncoderIn_read	616																																
To read data from an incremental encoder via DIO Class 1 input channels.																																	

DioCl1EncoderIn_setEncoderLines	617
To specify the number of encoder lines.	
DioCl1EncoderIn_setEncPosition	618
To specify the value of the current position in lines.	
DioCl1EncoderIn_setEncPosValidity	619
To specify the validity of the encoder position.	
DioCl1EncoderIn_setGatedMode	621
To specify the gated mode of the incremental encoder's IDX signal.	
DioCl1EncoderIn_setIndexMode	622
To specify how to react to an encoder index signal.	
DioCl1EncoderIn_setIndexPosition	623
To specify the position that is written to the position counter when an index signal pulse occurs.	
DioCl1EncoderIn_setInputFilter	624
To specify the time interval for the noise filtering of incremental encoder signals.	
DioCl1EncoderIn_setMaxPosition	626
To specify the encoder's maximum position value.	
DioCl1EncoderIn_setMeasurementInterval	627
To specify the measurement interval used to calculate the encoder's velocity.	
DioCl1EncoderIn_setMinEncVelocity	628
To specify the encoder's minimum velocity in lines per second.	
DioCl1EncoderIn_setMinMechVelocity	629
To specify the encoder's minimum velocity in revolutions per minute (rpm).	
DioCl1EncoderIn_setMinPosition	630
To specify the encoder's minimum position value.	
DioCl1EncoderIn_start	632
To start measuring incremental encoder signals via DIO Class 1 input channels.	
DioCl1EncoderIn_stop	633
To start measuring incremental encoder signals via DIO Class 1 input channels.	
DioCl1EncoderIn_write	634
To update the settings of incremental encoder signal measurement via DIO Class 1 input channels during run time.	

DioCl1EncoderIn_apply

Syntax

```
UInt32 DioCl1EncoderIn_apply(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn)
```

Include file

`IoDrvDioClass1EncoderIn.h`

Purpose

To apply the initialization settings to the DIO Class 1 incremental encoder channels.

Description

Before you can start to measure the incremental encoder signals via the specified DIO Class 1 channels, you have to transfer the settings of the `DioCl1EncoderIn` driver object to its related hardware by using `DioCl1EncoderIn_apply`.

You must do this also for the default values and for values that you specified via the `DioCl1EncoderIn_set<Parameter>` functions. The `DioCl1EncoderIn_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The signal measurement on the specified digital input channels is not activated until you have called `DioCl1EncoderIn_start`.

To update settings during run time, you must use `DioCl1EncoderIn_write`.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

`pDioCl1EncoderIn` Lets you specify the target incremental encoder via the corresponding `DioCl1EncoderIn` driver object. The `DioCl1EncoderIn` driver object had to be created before by using `DioCl1EncoderIn_create`.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics**References**

DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_setEncoderLines.....	617
DioCl1EncoderIn_setGatedMode.....	621
DioCl1EncoderIn_setIndexMode.....	622
DioCl1EncoderIn_setIndexPosition.....	623
DioCl1EncoderIn_setInputFilter.....	624
DioCl1EncoderIn_setMaxPosition.....	626
DioCl1EncoderIn_setMeasurementInterval.....	627
DioCl1EncoderIn_setMinPosition.....	630
DioCl1EncoderIn_start.....	632

DioCl1EncoderIn_create

Syntax

```
UInt32 DioCl1EncoderIn_create(
    DioCl1EncoderInSDrvObject **ppDioCl1EncIn,
    UInt32 Port,
    UInt32 Channel1,
    UInt32 Encoder,
    UInt32 IndexUsage)
```

Include file

IoDrvDioClass1EncoderIn.h

Purpose

To create the I/O driver object for incremental encoder signal measurement via DIO Class 1 input channels.

Description

This function is used to perform all the steps necessary to create an I/O driver object to access DIO Class 1 channels.

A DioCl1EncoderIn driver object handles one incremental encoder interface with input channels for the signals *PHI0*, *PHI90* and optionally *IDX*.

You allocate the required consecutive channels by specifying the first channel.

The incremental encoder I/O driver is initialized with default values so that it is ready for use.

The initial values are:

- Number of encoder lines: 256
- Input filter interval: 0 s
- Initial position: 0.00
- Index mode: Index signal not used
- Index position: 0.00

- Minimum position: -2,097,152.00
- Maximum position: 2,097,151.75
- Gated mode: Not gated
- Measurement interval: 1
- Position validity: Position invalid

The function is intended to be called during the initialization phase of the real-time application.

You can use the `DioCl1EncoderIn_set<Parameter>` functions to change the default parameters. Before you can start to measure incremental encoder signals, you have to transfer the settings of the DioCl1EncoderIn driver object to its related input channels by using `DioCl1EncoderIn_apply`.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

ppDioCl1Encln Lets you specify the address of a variable which holds the address of the created driver object for the incremental encoder.

Port Lets you specify the number of the DIO Class 1 port that provides the required channels in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
...	...
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

Channel Lets you specify the channel to be used for *PHIO* of the connected incremental encoder in the range 1 ... 15. The subsequent channel is used for *PHI90* and, if the index signal is used, the next channel for *IDX*. The channels used by the encoder must not be allocated by other DIO Class 1 functions.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified DIO Class 1 port.
...	...
DIO_CLASS1_CHANNEL_15	Specifies channel 15 of the specified DIO Class 1 port.

Encoder Lets you specify an instance number for the created driver object of the incremental encoder in the range 1 ... 6. One driver object can handle one incremental encoder interface. You can use the instance number to attach the encoder to an angle computation unit via `Acu_setInputSensor`.

Symbol	Meaning
DIO_ENC_INSTANCE_1	Specifies instance number 1.
...	...
DIO_ENC_INSTANCE_6	Specifies instance number 6.

You can specify up to six instance numbers for all drivers of incremental encoder interfaces.

IndexUsage Lets you specify whether the index signal (IDX) of the incremental encoder is used.

Symbol	Meaning
DIO_ENC_IUSAGE_DISABLED	Specify not to use the IDX signal of the encoder.
DIO_ENC_IUSAGE_ENABLED	Specify to use the IDX signal of the encoder.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_setEncoderLines.....	617
DioCl1EncoderIn_setGatedMode.....	621
DioCl1EncoderIn_setIndexMode.....	622
DioCl1EncoderIn_setIndexPosition.....	623
DioCl1EncoderIn_setInputFilter.....	624
DioCl1EncoderIn_setMaxPosition.....	626
DioCl1EncoderIn_setMeasurementInterval.....	627
DioCl1EncoderIn_setMinPosition.....	630
DioCl1EncoderIn_start.....	632

DioCl1EncoderIn_getEncPosition

Syntax

```
UInt32 DioCl1EncoderIn_getEncPosition(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    Float64 *pEncPosition)
```

Include file

IoDrvDioClass1EncoderIn.h

Purpose	To get the current encoder position in lines from the position counter.						
Description	<p>The encoder provides position counter values in the range that you can define via DioCl1EncoderIn_setMinPosition and DioCl1EncoderIn_setMaxPosition.</p> <p>By default, the incremental encoder only provides values relative to the position of the encoder at the last reset.</p> <p>To get absolute position values, you must use DioCl1EncoderIn_setIndexMode to enable the usage of the encoder's index signal (IDX). In this case, at the first IDX pulse, the encoder's position counter is set to the index position value. From then on DioCl1EncoderIn_getEncPosition provides absolute position values. You can also simulate an index signal with the functions DioCl1EncoderIn_setEncPosition and DioCl1EncoderIn_setEncPosValidity.</p> <p>You can specify the index position value via DioCl1EncoderIn_setIndexPosition.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call DioCl1EncoderIn_read to update the internal buffer with the latest values.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create.</p> <p>pEncPosition Lets you specify the address of the variable that holds the current encoder position in lines in the range -2,097,152.00 ... 2,097,151.75 with a resolution of 0.25 lines.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics

References

DioCl1EncoderIn_read.....	616
---------------------------	-----

DioCl1EncoderIn_getEncVelocity

Syntax

```
UInt32 DioCl1EncoderIn_getEncVelocity(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    Float64 *pEncVelocity)
```

Include file

IoDrvDioClass1EncoderIn.h

Purpose

To get the encoder velocity in lines per second.

Description

You can use **DioCl1EncoderIn_setMeasurementInterval** to specify how many measured values of the encoder are used to calculate its velocity.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl1EncoderIn_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using **DioCl1EncoderIn_create**.

pEncVelocity Lets you specify the address of the variable that holds the encoder velocity in lines per second in the range -10.0e6 ... +10.0e6 lines per second with a resolution of 0.187 lines per second.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1EncoderIn_read.....	616
---------------------------	-----

DioCl1EncoderIn_getIsIndexRaised

Syntax

```
UInt32 DioCl1EncoderIn_getIsIndexRaised(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    UInt32 *pIsIndexRaised)
```

Include file

`IoDrvDioClass1EncoderIn.h`

Purpose

To verify whether a first index signal pulse occurred.

Description

This function is relevant only if the following conditions are fulfilled:

- You specify to use the encoder's index signal (IDX) via **DioCl1EncoderIn_create**.
- You enable the IDX signal via **DioCl1EncoderIn_setIndexMode**.

The function is used to verify whether the encoder positions you get via **DioCl1EncoderIn_getEncPosition** and **DioCl1EncoderIn_getMechPosition** are absolute or relative values.

The incremental encoder provides relative position values until an IDX pulse occurs for the first time. Then the position counter is set to the index position value and absolute position values are provided.

You can specify the index position value via **DioCl1EncoderIn_setIndexPosition**.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl1EncoderIn_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Incremental Encoder Interface](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using [DioCl1EncoderIn_create](#).

plsIndexRaised Lets you specify the address of a variable that holds the result of the index raised check.

Value	Meaning
0	The index signal was not raised.
> 0	The index signal was raised.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

[DioCl1EncoderIn_read](#)..... 616

DioCl1EncoderIn_getMechPosition

Syntax

```
UInt32 DioCl1EncoderIn_getMechPosition(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    Float64 *pMechPosition)
```

Include file

[IoDrvDioClass1EncoderIn.h](#)

Purpose

To get the mechanical position of the encoder in degrees.

Description	The mechanical position is calculated from the current value of the encoder's position counter and the number of lines for a full revolution of the encoder. The number of lines is hardware-dependent and can be adjusted via DioCl1EncoderIn_setEncoderLines . The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call DioCl1EncoderIn_read to update the internal buffer with the latest values. If an error is detected, the first instance of the error is returned to the caller.
I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) . For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features) .
Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create.</p> <p>pMechPosition Lets you specify the address of the variable that holds the calculated encoder position in degrees in the range $0.0^\circ \dots (360^\circ \text{--resolution})$. The resolution equals $90/N^\circ$, where N is the number of encoder lines.</p>
Return value	The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References
	DioCl1EncoderIn_read 616

DioCl1EncoderIn_getMechVelocity

Syntax	<pre>UInt32 DioCl1EncoderIn_getMechVelocity(DioCl1EncoderInSDrvObject *pDioCl1EncoderIn, Float64 *pMechVelocity)</pre>
---------------	---

Include file	<code>IoDrvDioClass1EncoderIn.h</code>						
Purpose	To get the mechanical velocity of the encoder in revolutions per minute.						
Description	<p>If the number of encoder lines is set incorrectly, the calculated mechanical velocity will also be incorrect. You can change the number of lines via <code>DioCl1EncoderIn_setEncoderLines</code>.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call <code>DioCl1EncoderIn_read</code> to update the internal buffer with the latest values.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using <code>DioCl1EncoderIn_create</code>.</p> <p>pMechVelocity Lets you specify the address of the variable that holds the encoder velocity in the range $-6.0\text{e}8 \dots +6.0\text{e}8/N$ rpm, where N is the number of encoder lines. The resolution equals $11.19/N$ rpm.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						
Related topics	References						
	<table border="1"> <tr> <td>DioCl1EncoderIn_read.....</td><td>616</td></tr> </table>	DioCl1EncoderIn_read	616				
DioCl1EncoderIn_read	616						

DioCl1EncoderIn_read

Syntax	<pre>UInt32 DioCl1EncoderIn_read(DioCl1EncoderInSDrvObject *pDioCl1EncoderIn)</pre>
Include file	<code>IoDrvDioClass1EncoderIn.h</code>
Purpose	To read data from an incremental encoder via DIO Class 1 input channels.
Description	<p>This function is used to read the raw data of the DIO Class 1 channels that are connected to the incremental encoder. Then it evaluates the sensor position and velocity. It stores the results as a consistent data set in an internal buffer.</p> <p>The input channels must be enabled before via DioCl1EncoderIn_start.</p> <p>To get the values from the internal buffer, you must use one of the following functions:</p> <ul style="list-style-type: none"> ▪ DioCl1EncoderIn_getEncPosition to get the sensor position in lines ▪ DioCl1EncoderIn_getEncVelocity to get the sensor velocity in lines per second ▪ DioCl1EncoderIn_getMechPosition to get the mechanical position of the rotor in degrees ▪ DioCl1EncoderIn_getMechVelocity to get the mechanical velocity of the rotor in revolutions per minute ▪ DioCl1EncoderIn_getIsIndexRaised to verify whether a first index signal pulse occurred <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>
Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1EncoderIn_getEncPosition.....	609
DioCl1EncoderIn_getEncVelocity.....	611
DioCl1EncoderIn_getIsIndexRaised.....	612
DioCl1EncoderIn_getMechPosition.....	613
DioCl1EncoderIn_getMechVelocity.....	614
DioCl1EncoderIn_start.....	632

DioCl1EncoderIn_setEncoderLines

Syntax

```
UInt32 DioCl1EncoderIn_setEncoderLines(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    UInt32 EncoderLines)
```

Include file

`IoDrvDioClass1EncoderIn.h`

Purpose

To specify the number of encoder lines.

Description

To get this hardware-dependent information, refer to the documentation of the encoder's manufacturer.

These settings do not take effect until you call `DioCl1EncoderIn_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create.</p> <p>EncoderLines Lets you specify the encoder's number of lines in the range 1 ... 2^{21}. The default value is 256.</p>
Return value	The function returns an error code.

Related topics	References						
	<table> <tr> <td>DioCl1EncoderIn_apply.....</td> <td>606</td> </tr> <tr> <td>DioCl1EncoderIn_create.....</td> <td>607</td> </tr> <tr> <td>DioCl1EncoderIn_getEncPosition.....</td> <td>609</td> </tr> </table>	DioCl1EncoderIn_apply	606	DioCl1EncoderIn_create	607	DioCl1EncoderIn_getEncPosition	609
DioCl1EncoderIn_apply	606						
DioCl1EncoderIn_create	607						
DioCl1EncoderIn_getEncPosition	609						

DioCl1EncoderIn_setEncPosition

Syntax	<pre>UInt32 DioCl1EncoderIn_setEncPosition(DioCl1EncoderInSDrvObject *pDioCl1EncoderIn, Float64 Position)</pre>
Include file	<code>IoDrvDioClass1EncoderIn.h</code>
Purpose	To specify the value of the current encoder position in lines.
Description	<p>The encoder position that you specify must be in the defined position range. You can change the position range via DioCl1EncoderIn_setMinPosition and DioCl1EncoderIn_setMaxPosition.</p> <p>The setting does not take effect until you call DioCl1EncoderIn_apply during the initialization phase or DioCl1EncoderIn_write during run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.</p>

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Incremental Encoder Interface](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using [DioCl1EncoderIn_create](#).

Position Lets you specify the current encoder position in lines in the range -2,097,152.00 ... 2,097,151.75 with a resolution of 0.25 lines. The default position is 0.00.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_setMaxPosition.....	626
DioCl1EncoderIn_setMinPosition.....	630
DioCl1EncoderIn_start.....	632
DioCl1EncoderIn_write.....	634

DioCl1EncoderIn_setEncPosValidity

Syntax

```
UInt32 DioCl1EncoderIn_setEncPosValidity(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    UInt32 ValidityState)
```

Include file

`IoDrvDioClass1EncoderIn.h`

Purpose

To specify the validity of the encoder position.

Description	<p>The validity information of the encoder position is relevant if the encoder interface is connected to the angle computation unit (ACU). The ACU evaluates this validity information. Typically only valid encoder positions are used by the ACU. If the ACU receives invalid values, the ACU automatically switches to the sensor that provides valid values.</p> <p>DioCl1EncoderIn_setEncPosValidity is useful if the encoder does not provide an index signal.</p> <p>The setting does not take effect until you call DioCl1EncoderIn_apply during the initialization phase or DioCl1EncoderIn_write during run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.</p>								
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>								
Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create.</p> <p>ValidityState Lets you specify the validity of the encoder position.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_ENC_POSITION_INVALID</td><td>Specifies the encoder position as invalid</td></tr> <tr> <td>DIO_ENC_POSITION_VALID</td><td>Specifies the encoder position as valid</td></tr> </tbody> </table>	Symbol	Meaning	DIO_ENC_POSITION_INVALID	Specifies the encoder position as invalid	DIO_ENC_POSITION_VALID	Specifies the encoder position as valid		
Symbol	Meaning								
DIO_ENC_POSITION_INVALID	Specifies the encoder position as invalid								
DIO_ENC_POSITION_VALID	Specifies the encoder position as valid								
Return value	The function returns an error code.								
Related topics	<p>References</p> <table border="1"> <tbody> <tr> <td>DioCl1EncoderIn_apply.....</td><td>606</td></tr> <tr> <td>DioCl1EncoderIn_setEncPosition.....</td><td>618</td></tr> <tr> <td>DioCl1EncoderIn_start.....</td><td>632</td></tr> <tr> <td>DioCl1EncoderIn_write.....</td><td>634</td></tr> </tbody> </table>	DioCl1EncoderIn_apply.....	606	DioCl1EncoderIn_setEncPosition.....	618	DioCl1EncoderIn_start.....	632	DioCl1EncoderIn_write.....	634
DioCl1EncoderIn_apply.....	606								
DioCl1EncoderIn_setEncPosition.....	618								
DioCl1EncoderIn_start.....	632								
DioCl1EncoderIn_write.....	634								

DioCl1EncoderIn_setGatedMode

Syntax

```
UInt32 DioCl1EncoderIn_setGatedMode(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    UInt32 GatedMode)
```

Include file

`IoDrvDioClass1EncoderIn.h`

Purpose

To specify the gated mode of the incremental encoder's IDX signal.

Description

This function is relevant only if you want to use the encoder's index signal (IDX).

The function is used to specify whether the *IDX* is gated to the *PHI0* and *PHI90* signals. This means that, the *IDX* signal is ignored unless *PHI0* and *PHI90* are both *High*.

These settings do not take effect until you call `DioCl1EncoderIn_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using `DioCl1EncoderIn_create`.

GatedMode Lets you specify whether the IDX signal is gated.

Symbol	Meaning
<code>DIO_ENC_GMODE_DISABLED</code>	Disables the gated mode of the incremental encoder's IDX signal (default).
<code>DIO_ENC_GMODE_ENABLED</code>	Enables the gated mode of the incremental encoder's IDX signal.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics**References**

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_setIndexMode.....	622
DioCl1EncoderIn_setIndexPosition.....	623

DioCl1EncoderIn_setIndexMode

Syntax

```
UInt32 DioCl1EncoderIn_setIndexMode(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    UInt32 IndexMode)
```

Include file

IoDrvDioClass1EncoderIn.h

Purpose

To specify how to react to an encoder index signal.

Description

This function is relevant only if you want to use the encoder's index signal (IDX).

The function is used to reset the encoder's position counter to the specified index position either at each index pulse or only at the first index pulse.

The default index position is 0.00. You can change its value via **DioCl1EncoderIn_setIndexPosition**.

These settings do not take effect until you call **DioCl1EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using **DioCl1EncoderIn_create**.

IndexMode Lets you specify how to react to an IDX signal.

Symbol	Meaning
DIO_ENC_IMODE_DISABLED	Specifies not to use the IDX signal (default).
DIO_ENC_EVERY_INDEX	Specifies to reset the position counter at every IDX pulse.
DIO_ENC_FIRST_INDEX	Specifies to reset the position counter only at the first IDX pulse.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_setGatedMode.....	621
DioCl1EncoderIn_setIndexPosition.....	623

DioCl1EncoderIn_setIndexPosition

Syntax

```
UInt32 DioCl1EncoderIn_setIndexPosition(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    Float64 IndexPos)
```

Include file

IoDrvDioClass1EncoderIn.h

Purpose

To specify the position that is written to the position counter when an index signal pulse occurs.

Description

This function is relevant only if the following conditions are fulfilled:

- You specify to use the encoder's index signal (IDX) via **DioCl1EncoderIn_create**.
- You enable the IDX signal via **DioCl1EncoderIn_setIndexMode**.

The index position that you specify must be in the defined position range. You can change the position range via **DioCl1EncoderIn_setMinPosition** and **DioCl1EncoderIn_setMaxPosition**.

These settings do not take effect until you call **DioCl1EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using **DioCl1EncoderIn_create**.

IndexPos Lets you specify the index position in lines in the range -2,097,152.00 ... 2,097,151.75 with a resolution of 0.25 lines. The default index position is 0.00.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_setIndexMode.....	622
DioCl1EncoderIn_setMaxPosition.....	626
DioCl1EncoderIn_setMinPosition.....	630

[DioCl1EncoderIn_setInputFilter](#)**Syntax**

```
UInt32 DioCl1EncoderIn_setInputFilter(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    Float64 FilterInterval)
```

Include file	<code>IoDrvDioClass1EncoderIn.h</code>						
Purpose	To specify the time interval for the noise filtering of incremental encoder signals on DIO Class 1 input channels.						
Description	<p>Only those input signals will be processed which have constant values over a period of time longer than the FilterInterval parameter specifies.</p> <p>The input filter will be applied to all the channels that are controlled by the specified DioCl1EncoderIn driver object.</p> <p>These settings do not take effect until you call DioCl1EncoderIn_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create.</p> <p>FilterInterval Lets you specify the minimum time interval for which the input signal(s) must provide a constant value that is to be used for processing. Shorter input signals will be filtered. You can specify filter intervals in seconds in the range 0 ... 0.01 s.</p> <p>The resolution/deviation depends on the specified range:</p> <ul style="list-style-type: none"> ▪ 0 ... 640 ns Resolution of 10 ns ▪ >640 ns ... 0.01 s Deviation of <10% 						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics**References**

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607

DioCl1EncoderIn_setMaxPosition

Syntax

```
UInt32 DioCl1EncoderIn_setMaxPosition(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    Float64 MaximumPos)
```

Include file

IoDrvDioClass1EncoderIn.h

Purpose

To specify the encoder's maximum position value.

Description

This function is used to specify the maximum position value in lines for the encoder's position counter.

If the encoder counts forwards and its value reaches the maximum position value, the position counter is set to the minimum position value that you specified via **DioCl1EncoderIn_setMinPosition**.

If the encoder counts backwards and its value reaches the minimum position value, the position counter is set to the maximum position value.

These settings do not take effect until you call **DioCl1EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using **DioCl1EncoderIn_create**.

MaximumPos Lets you specify the maximum position in lines in the range -2,097,152.00 ... 2,097,151.75 lines with a resolution of 0.25 lines. The default value for the maximum position is +2,097,151.75 lines.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_getEncPosition.....	609
DioCl1EncoderIn_setMinPosition.....	630

DioCl1EncoderIn_setMeasurementInterval

Syntax

```
UInt32 DioCl1EncoderIn_SetMeasurementInterval(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    UInt32 MeasInterval)
```

Include file

IoDrvDioClass1EncoderIn.h

Purpose

To specify the measurement interval used to calculate the encoder's velocity.

Description

This function is used to specify how many measured values of the encoder are used to calculate its velocity. For example, if you specify a measurement interval of three, the velocity calculation is based on the average of the encoder data captured by the last three calls of **DioCl1EncoderIn_read**.

These settings do not take effect until you call **DioCl1EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration ).								
	For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features ).								
Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create.</p> <p>MeasInterval Lets you specify the number of measurement intervals in the range 1 ... 20. The default value is 1.</p>								
Return value	The function returns an error code.								
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.		
Error Code	Meaning								
IOLIB_NO_ERROR	The function was successfully completed.								
!= IOLIB_NO_ERROR	The function was not completed.								
Related topics	References								
	<table> <tbody> <tr> <td>DioCl1EncoderIn_apply.....</td><td>606</td></tr> <tr> <td>DioCl1EncoderIn_create.....</td><td>607</td></tr> <tr> <td>DioCl1EncoderIn_getEncVelocity.....</td><td>611</td></tr> <tr> <td>DioCl1EncoderIn_read.....</td><td>616</td></tr> </tbody> </table>	DioCl1EncoderIn_apply.....	606	DioCl1EncoderIn_create.....	607	DioCl1EncoderIn_getEncVelocity.....	611	DioCl1EncoderIn_read.....	616
DioCl1EncoderIn_apply.....	606								
DioCl1EncoderIn_create.....	607								
DioCl1EncoderIn_getEncVelocity.....	611								
DioCl1EncoderIn_read.....	616								

DioCl1EncoderIn_setMinEncVelocity

Syntax	<pre>UInt32 DioCl1EncoderIn_setMinEncVelocity(DioCl1EncoderInSDrvObject *pDioCl1EncoderIn, Float64 MinEncVelocity)</pre>
Include file	<code>IoDrvDioClass1EncoderIn.h</code>
Purpose	To specify the encoder's minimum velocity in lines per second.
Description	This function is used to specify the minimum velocity to be measured. If the velocity falls below the specified minimum value, the measured velocity is set to 0, representing a standstill detection of the motor.

These settings do not take effect until you call **DioCl1EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Incremental Encoder Interface](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using **DioCl1EncoderIn_create**.

MinEncVelocity Lets you specify the minimum encoder velocity in lines in the range 1.0 ... 1.0e+6 lines per second.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_setMinMechVelocity.....	629

DioCl1EncoderIn_setMinMechVelocity

Syntax

```
UInt32 DioCl1EncoderIn_setMinMechVelocity(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn,
    Float64 MinMechVelocity)
```

Include file

`IoDrvDioClass1EncoderIn.h`

Purpose

To specify the encoder's minimum velocity in revolutions per minute (rpm).

Description	This function is used to specify the minimum velocity to be measured. If the velocity falls below the specified minimum value, the measured velocity is set to 0, representing a standstill detection of the motor. These settings do not take effect until you call DioCl1EncoderIn_apply during the initialization phase. If an error is detected, the first instance of the error is returned to the caller and an error message is output.
I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) . For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features) .
Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create.</p> <p>MinMechVelocity Lets you specify the minimum encoder velocity in revolutions per minute in the range 0.01 ... 10,000.0 rpm.</p>
Return value	The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References						
	<table> <tr> <td>DioCl1EncoderIn_apply.....</td> <td>606</td> </tr> <tr> <td>DioCl1EncoderIn_create.....</td> <td>607</td> </tr> <tr> <td>DioCl1EncoderIn_setMinEncVelocity.....</td> <td>628</td> </tr> </table>	DioCl1EncoderIn_apply.....	606	DioCl1EncoderIn_create.....	607	DioCl1EncoderIn_setMinEncVelocity.....	628
DioCl1EncoderIn_apply.....	606						
DioCl1EncoderIn_create.....	607						
DioCl1EncoderIn_setMinEncVelocity.....	628						

DioCl1EncoderIn_setMinPosition

Syntax	<pre>UInt32 DioCl1EncoderIn_setMinPosition(DioCl1EncoderInSDrvObject *pDioCl1EncoderIn, Float64 MinimumPos)</pre>
---------------	--

Include file	<code>IoDrvDioClass1EncoderIn.h</code>						
Purpose	To specify the encoder's minimum position value.						
Description	<p>This function is used to specify the minimum position value in lines for the encoder's position counter.</p> <p>If the encoder counts backwards and its value reaches the minimum position value, the position counter is set to the maximum position value that you specified via DioCl1EncoderIn_setMaxPosition.</p> <p>If the encoder counts forwards and its value reaches the maximum position value, the position counter is set to the minimum position value.</p> <p>These settings do not take effect until you call DioCl1EncoderIn_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create.</p> <p>MinimumPos Lets you specify the minimum position in lines in the range -2,097,152.00 ... 2,097,151.75 with a resolution of 0.25 lines. The default value for the minimum position is -2,097,152.00 lines.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	References						
	<table border="0"> <tr> <td style="padding-right: 20px;">DioCl1EncoderIn_apply.....</td><td>606</td></tr> <tr> <td>DioCl1EncoderIn_create.....</td><td>607</td></tr> </table>	DioCl1EncoderIn_apply	606	DioCl1EncoderIn_create	607		
DioCl1EncoderIn_apply	606						
DioCl1EncoderIn_create	607						

DioCl1EncoderIn_getEncPosition.....	609
DioCl1EncoderIn_setMaxPosition.....	626

DioCl1EncoderIn_start

Syntax

```
UInt32 DioCl1EncoderIn_start(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn)
```

Include file

`IoDrvDioClass1EncoderIn.h`

Purpose

To start measuring incremental encoder signals via DIO Class 1 input channels.

Description

This function is used to start the specified DioCl1EncoderIn driver so that it will activate its digital input channels. The digital input channels are enabled to start capturing the incremental encoder input signals.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using [DioCl1EncoderIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_read.....	616
DioCl1EncoderIn_write.....	634

DioCl1EncoderIn_stop

Syntax

```
UInt32 DioCl1EncoderIn_stop(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn)
```

Include file

IoDrvDioClass1EncoderIn.h

Purpose

To disable the driver object to measure incremental encoder signals via DIO Class 1 input channels.

Description

This function is used to stop the specified DioCl1EncoderIn driver so that it will deactivate its digital input channels. The digital input channels are disabled to stop capturing the incremental encoder input signals.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using [DioCl1EncoderIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_read.....	616
DioCl1EncoderIn_write.....	634

DioCl1EncoderIn_write

Syntax

```
UInt32 DioCl1EncoderIn_write(
    DioCl1EncoderInSDrvObject *pDioCl1EncoderIn)
```

Include file

`IoDrvDioClass1EncoderIn.h`

Purpose

To update the settings of incremental encoder signal measurement via DIO Class 1 input channels during run time.

Description

If you have used the following function during run time, its settings will not be updated on the hardware until you call `DioCl1EncoderIn_write`:

- `DioCl1EncoderIn_setEncPosition`
- `DioCl1EncoderIn_setEncPosValidity`

The specified digital input channels must be enabled before by using `DioCl1EncoderIn_start`.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters	pDioCl1EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl1EncoderIn driver object. The DioCl1EncoderIn driver object had to be created before by using DioCl1EncoderIn_create .
-------------------	--

Return value	The function returns an error code.
---------------------	-------------------------------------

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_create.....	607
DioCl1EncoderIn_setEncPosition.....	618
DioCl1EncoderIn_start.....	632

Incremental Encoder Class 2

Introduction

Incremental encoders that are connected to DIO Class 2 channels can be accessed via the **DioCl2EncoderIn** functions.

Where to go from here**Information in this section**

DioCl2EncoderIn_apply.....	637
To apply the initialization settings to the DIO Class 2 incremental encoder channels.	
DioCl2EncoderIn_create.....	638
To create the I/O driver object for incremental encoder signal measurement via DIO Class 2 input channels.	
DioCl2EncoderIn_getEncPosition.....	640
To get the current encoder position in lines from the position counter.	
DioCl2EncoderIn_getEncVelocity.....	642
To get the encoder velocity in lines per second.	
DioCl2EncoderIn_getIsIndexRaised.....	643
To verify whether a first index signal pulse occurred.	

DioCl2EncoderIn_getMechPosition	644
To get the mechanical position of the encoder in degrees.	
DioCl2EncoderIn_getMechVelocity	645
To get the mechanical velocity of the encoder in revolutions per minute.	
DioCl2EncoderIn_read	647
To read data from an incremental encoder via DIO Class 2 input channels.	
DioCl2EncoderIn_setEncoderLines	648
To specify the number of encoder lines.	
DioCl2EncoderIn_setEncPosition	649
To specify the value of the current position in lines.	
DioCl2EncoderIn_setEncPosValidity	650
To specify the validity of the encoder position.	
DioCl2EncoderIn_setGatedMode	652
To specify the gated mode of the incremental encoder's IDX signal.	
DioCl2EncoderIn_setIndexMode	653
To specify how to react to an encoder index signal.	
DioCl2EncoderIn_setIndexPosition	654
To specify the position that is written to the position counter when an index signal pulse occurs.	
DioCl2EncoderIn_setInputFilter	655
To specify the time interval for the noise filtering of incremental encoder signals.	
DioCl2EncoderIn_setMaxPosition	657
To specify the encoder's maximum position value.	
DioCl2EncoderIn_setMeasurementInterval	658
To specify the measurement interval used to calculate the encoder's velocity.	
DioCl2EncoderIn_setMinEncVelocity	659
To specify the encoder's minimum velocity in lines per second.	
DioCl2EncoderIn_setMinMechVelocity	660
To specify the encoder's minimum velocity in revolutions per minute (rpm).	
DioCl2EncoderIn_setMinPosition	661
To specify the encoder's minimum position value.	
DioCl2EncoderIn_start	663
To start measuring incremental encoder signals via DIO Class 2 input channels.	
DioCl2EncoderIn_stop	664
To start measuring incremental encoder signals via DIO Class 2 input channels.	

DioCl2EncoderIn_write.....665

To update the settings of incremental encoder signal measurement via DIO Class 2 input channels during run time.

DioCl2EncoderIn_apply

Syntax

```
UInt32 DioCl2EncoderIn_apply(  
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn)
```

Include file

IoDrvDioClass2EncoderIn.h

Purpose

To apply the initialization settings to the DIO Class 2 incremental encoder channels.

Description

Before you can start to measure the incremental encoder signals via the specified DIO Class 2 channels, you have to transfer the settings of the DioCl2EncoderIn driver object to its related hardware by using **DioCl2EncoderIn_apply**.

You must do this also for the default values and for values that you specified via the **DioCl2EncoderIn_set<Parameter>** functions. The **DioCl2EncoderIn_apply** function is intended to be called at the end of the initialization phase of the real-time application.

The signal measurement on the specified digital input channels is not activated until you have called **DioCl2EncoderIn_start**.

To update settings during run time, you must use **DioCl2EncoderIn_write**.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using **DioCl2EncoderIn_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_setEncoderLines.....	648
DioCl2EncoderIn_setGatedMode.....	652
DioCl2EncoderIn_setIndexMode.....	653
DioCl2EncoderIn_setIndexPosition.....	654
DioCl2EncoderIn_setInputFilter.....	655
DioCl2EncoderIn_setMaxPosition.....	657
DioCl2EncoderIn_setMeasurementInterval.....	658
DioCl2EncoderIn_setMinPosition.....	661
DioCl2EncoderIn_start.....	663

DioCl2EncoderIn_create

Syntax

```
UInt32 DioCl2EncoderIn_create(
    DioCl2EncoderInSDrvObject **ppDioCl2EncIn,
    UInt32 Channel,
    UInt32 Encoder,
    UInt32 IndexUsage)
```

Include file

IoDrvDioClass2EncoderIn.h

Purpose

To create the I/O driver object for incremental encoder signal measurement via DIO Class 2 input channels.

Description

This function is used to perform all the steps necessary to create an I/O driver object to access DIO Class 2 channels.

A DioCl2EncoderIn driver object handles one incremental encoder interface with input channels for the signals *PHI0*, *PHI90* and optionally *IDX*.

You allocate the required consecutive channels by specifying the first channel.

The incremental encoder I/O driver is initialized with default values so that it is ready for use.

The initial values are:

- Number of encoder lines: 256
- Input filter interval: 0 s
- Initial position: 0.00
- Index mode: Index signal not used
- Index position: 0.00
- Minimum position: -2,097,152.00
- Maximum position: 2,097,151.75
- Gated mode: Not gated
- Measurement interval: 1
- Position validity: Position invalid

The function is intended to be called during the initialization phase of the real-time application.

You can use the `DioCl2EncoderIn_set<Parameter>` functions to change the default parameters. Before you can start to measure incremental encoder signals, you have to transfer the settings of the `DioCl2EncoderIn` driver object to its related input channels by using `DioCl2EncoderIn_apply`.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

ppDioCl2Encln Lets you specify the address of a variable which holds the address of the created driver object for the incremental encoder.

Channel Lets you specify the channel to be used for `PHI0` of the connected incremental encoder in the range 1 ... 11. The subsequent channel is used for `PHI90` and, if the index signal is used, the next channel for `IDX`. The channels used by the encoder must not be allocated by other DIO Class 2 functions.

Symbol	Meaning
DIO_CLASS2_CHANNEL_1	Specifies channel 1 of DIO Class 2.
...	...
DIO_CLASS2_CHANNEL_11	Specifies channel 11 of DIO Class 2.

Encoder Lets you specify an instance number for the created driver object of the incremental encoder in the range 1 ... 6. One driver object can handle one incremental encoder interface. You can use the instance number to attach the encoder to an angle computation unit via `Acu_setInputSensor`.

Symbol	Meaning
DIO_ENC_INSTANCE_1	Specifies instance number 1.
...	...
DIO_ENC_INSTANCE_6	Specifies instance number 6.

You can specify up to six instance numbers for all drivers of incremental encoder interfaces.

IndexUsage Lets you specify whether the index signal (IDX) of the incremental encoder is used.

Symbol	Meaning
DIO_ENC_IUSAGE_DISABLED	Specify not to use the IDX signal of the encoder.
DIO_ENC_IUSAGE_ENABLED	Specify to use the IDX signal of the encoder.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_setEncoderLines.....	648
DioCl2EncoderIn_setGatedMode.....	652
DioCl2EncoderIn_setIndexMode.....	653
DioCl2EncoderIn_setIndexPosition.....	654
DioCl2EncoderIn_setInputFilter.....	655
DioCl2EncoderIn_setMaxPosition.....	657
DioCl2EncoderIn_setMeasurementInterval.....	658
DioCl2EncoderIn_setMinPosition.....	661
DioCl2EncoderIn_start.....	663

DioCl2EncoderIn_getEncPosition

Syntax

```
UInt32 DioCl2EncoderIn_getEncPosition(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    Float64 *pEncPosition)
```

Include file

IoDrvDioClass2EncoderIn.h

Purpose	To get the current encoder position in lines from the position counter.						
Description	<p>The encoder provides position counter values in the range that you can define via DioCl2EncoderIn_setMinPosition and DioCl2EncoderIn_setMaxPosition.</p> <p>By default, the incremental encoder only provides values relative to the position of the encoder at the last reset.</p> <p>To get absolute position values, you must use DioCl2EncoderIn_setIndexMode to enable the usage of the encoder's index signal (IDX). In this case, at the first IDX pulse, the encoder's position counter is set to the index position value. From then on DioCl2EncoderIn_getEncPosition provides absolute position values. You can also simulate an index signal with the functions DioCl2EncoderIn_setEncPosition and DioCl2EncoderIn_setEncPosValidity.</p> <p>You can specify the index position value via DioCl2EncoderIn_setIndexPosition.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call DioCl2EncoderIn_read to update the internal buffer with the latest values.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using DioCl2EncoderIn_create.</p> <p>pEncPosition Lets you specify the address of the variable that holds the current encoder position in lines in the range -2,097,152.00 ... 2,097,151.75 with a resolution of 0.25 lines.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics**References**

DioCl2EncoderIn_read	647
--	-----

DioCl2EncoderIn_getEncVelocity

Syntax

```
UInt32 DioCl2EncoderIn_getEncVelocity(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    Float64 *pEncVelocity)
```

Include file

IoDrvDioClass2EncoderIn.h

Purpose

To get the encoder velocity in lines per second.

Description

You can use **DioCl2EncoderIn_setMeasurementInterval** to specify how many measured values of the encoder are used to calculate its velocity.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl2EncoderIn_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using **DioCl2EncoderIn_create**.

pEncVelocity Lets you specify the address of the variable that holds the encoder velocity in lines per second in the range -10.0e6 ... +10.0e6 lines per second with a resolution of 0.187 lines per second.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_read.....	647
---------------------------	-----

DioCl2EncoderIn_getIsIndexRaised

Syntax

```
UInt32 DioCl2EncoderIn_getIsIndexRaised(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    UInt32 *pIsIndexRaised)
```

Include file

`IoDrvDioClass2EncoderIn.h`

Purpose

To verify whether a first index signal pulse occurred.

Description

This function is relevant only if the following conditions are fulfilled:

- You specify to use the encoder's index signal (IDX) via **DioCl2EncoderIn_create**.
- You enable the IDX signal via **DioCl2EncoderIn_setIndexMode**.

The function is used to verify whether the encoder positions you get via **DioCl2EncoderIn_getEncPosition** and **DioCl2EncoderIn_getMechPosition** are absolute or relative values.

The incremental encoder provides relative position values until an IDX pulse occurs for the first time. Then the position counter is set to the index position value and absolute position values are provided.

You can specify the index position value via **DioCl2EncoderIn_setIndexPosition**.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl2EncoderIn_read** to update the internal buffer with the latest values.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Incremental Encoder Interface](#) ([MicroLabBox Features](#)).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using [DioCl2EncoderIn_create](#).

plsIndexRaised Lets you specify the address of a variable that holds the result of the index raised check.

Value	Meaning
0	The index signal was not raised.
> 0	The index signal was raised.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

[DioCl2EncoderIn_read](#)..... 647

DioCl2EncoderIn_getMechPosition

Syntax

```
UInt32 DioCl2EncoderIn_getMechPosition(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    Float64 *pMechPosition)
```

Include file

[IoDrvDioClass2EncoderIn.h](#)

Purpose

To get the mechanical position of the encoder in degrees.

Description	The mechanical position is calculated from the current value of the encoder's position counter and the number of lines for a full revolution of the encoder. The number of lines is hardware-dependent and can be adjusted via DioCl2EncoderIn_setEncoderLines . The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call DioCl2EncoderIn_read to update the internal buffer with the latest values. If an error is detected, the first instance of the error is returned to the caller.
I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) . For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features) .
Parameters	<p>pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using DioCl2EncoderIn_create.</p> <p>pMechPosition Lets you specify the address of the variable that holds the calculated encoder position in degrees in the range $0.0^\circ \dots (360^\circ - \text{resolution})$. The resolution equals $90/N^\circ$, where N is the number of encoder lines.</p>
Return value	The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References
	DioCl2EncoderIn_read 647

DioCl2EncoderIn_getMechVelocity

Syntax	<pre>UInt32 DioCl2EncoderIn_getMechVelocity(DioCl2EncoderInSDrvObject *pDioCl2EncoderIn, Float64 *pMechVelocity)</pre>
---------------	---

Include file	<code>IoDrvDioClass2EncoderIn.h</code>						
Purpose	To get the mechanical velocity of the encoder in revolutions per minute.						
Description	<p>If the number of encoder lines is set incorrectly, the calculated mechanical velocity will also be incorrect. You can change the number of lines via <code>DioCl2EncoderIn_setEncoderLines</code>.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call <code>DioCl2EncoderIn_read</code> to update the internal buffer with the latest values.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using <code>DioCl2EncoderIn_create</code>.</p> <p>pMechVelocity Lets you specify the address of the variable that holds the encoder velocity in the range $-6.0\text{e}8 \dots +6.0\text{e}8/N$ rpm, where N is the number of encoder lines. The resolution equals $11.19/N$ rpm.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>\neq IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	\neq IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
\neq IOLIB_NO_ERROR	The function was not completed.						
Related topics	References						
	<table border="1"> <tr> <td>DioCl2EncoderIn_read.....</td><td>647</td></tr> </table>	DioCl2EncoderIn_read	647				
DioCl2EncoderIn_read	647						

DioCl2EncoderIn_read

Syntax

```
UInt32 DioCl2EncoderIn_read(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn)
```

Include file

`IoDrvDioClass2EncoderIn.h`

Purpose

To read data from an incremental encoder via DIO Class 2 input channels.

Description

This function is used to read the raw data of the DIO Class 2 channels that are connected to the incremental encoder. Then it evaluates the sensor position and velocity. It stores the results as a consistent data set in an internal buffer.

The input channels must be enabled before via [DioCl2EncoderIn_start](#).

To get the values from the internal buffer, you must use one of the following functions:

- [DioCl2EncoderIn_getEncPosition](#) to get the sensor position in lines
- [DioCl2EncoderIn_getEncVelocity](#) to get the sensor velocity in lines per second
- [DioCl2EncoderIn_getMechPosition](#) to get the mechanical position of the rotor in degrees
- [DioCl2EncoderIn_getMechVelocity](#) to get the mechanical velocity of the rotor in revolutions per minute
- [DioCl2EncoderIn_getIsIndexRaised](#) to verify whether a first index signal pulse occurred

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [Incremental Encoder Interface](#) ([MicroLabBox Features](#) ).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using [DioCl2EncoderIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_getEncPosition.....	640
DioCl2EncoderIn_getEncVelocity.....	642
DioCl2EncoderIn_getIsIndexRaised.....	643
DioCl2EncoderIn_getMechPosition.....	644
DioCl2EncoderIn_getMechVelocity.....	645
DioCl2EncoderIn_start.....	663

DioCl2EncoderIn_setEncoderLines

Syntax

```
UInt32 DioCl2EncoderIn_setEncoderLines(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    UInt32 EncoderLines)
```

Include file

`IoDrvDioClass2EncoderIn.h`

Purpose

To specify the number of encoder lines.

Description

To get this hardware-dependent information, refer to the documentation of the encoder's manufacturer.

These settings do not take effect until you call `DioCl2EncoderIn_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters	<p>pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using DioCl2EncoderIn_create.</p> <p>EncoderLines Lets you specify the encoder's number of lines in the range 1 ... 2^{21}. The default value is 256.</p>
Return value	The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References						
<table> <tr> <td>DioCl2EncoderIn_apply.....</td> <td>637</td> </tr> <tr> <td>DioCl2EncoderIn_create.....</td> <td>638</td> </tr> <tr> <td>DioCl2EncoderIn_getEncPosition.....</td> <td>640</td> </tr> </table>		DioCl2EncoderIn_apply	637	DioCl2EncoderIn_create	638	DioCl2EncoderIn_getEncPosition	640
DioCl2EncoderIn_apply	637						
DioCl2EncoderIn_create	638						
DioCl2EncoderIn_getEncPosition	640						

DioCl2EncoderIn_setEncPosition

Syntax	<pre>UInt32 DioCl2EncoderIn_setEncPosition(DioCl2EncoderInSDrvObject *pDioCl2EncoderIn, Float64 Position)</pre>
Include file	<code>IoDrvDioClass2EncoderIn.h</code>
Purpose	To specify the value of the current encoder position in lines.
Description	<p>The encoder position that you specify must be in the defined position range. You can change the position range via DioCl2EncoderIn_setMinPosition and DioCl2EncoderIn_setMaxPosition.</p> <p>The setting does not take effect until you call DioCl2EncoderIn_apply during the initialization phase or DioCl2EncoderIn_write during run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.</p>

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration ).
	For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features ).
Parameters	<p>pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using DioCl2EncoderIn_create.</p> <p>Position Lets you specify the current encoder position in lines in the range -2,097,152.00 ... 2,097,151.75 with a resolution of 0.25 lines. The default position is 0.00.</p>
Return value	The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References										
<table border="0"> <tr> <td>DioCl2EncoderIn_apply.....</td> <td>637</td> </tr> <tr> <td>DioCl2EncoderIn_setMaxPosition.....</td> <td>657</td> </tr> <tr> <td>DioCl2EncoderIn_setMinPosition.....</td> <td>661</td> </tr> <tr> <td>DioCl2EncoderIn_start.....</td> <td>663</td> </tr> <tr> <td>DioCl2EncoderIn_write.....</td> <td>665</td> </tr> </table>		DioCl2EncoderIn_apply.....	637	DioCl2EncoderIn_setMaxPosition.....	657	DioCl2EncoderIn_setMinPosition.....	661	DioCl2EncoderIn_start.....	663	DioCl2EncoderIn_write.....	665
DioCl2EncoderIn_apply.....	637										
DioCl2EncoderIn_setMaxPosition.....	657										
DioCl2EncoderIn_setMinPosition.....	661										
DioCl2EncoderIn_start.....	663										
DioCl2EncoderIn_write.....	665										

DioCl2EncoderIn_setEncPosValidity

Syntax	<pre>UInt32 DioCl2EncoderIn_setEncPosValidity(DioCl2EncoderInSDrvObject *pDioCl2EncoderIn, UInt32 ValidityState)</pre>
Include file	<code>IoDrvDioClass2EncoderIn.h</code>
Purpose	To specify the validity of the encoder position.

Description	<p>The validity information of the encoder position is relevant if the encoder interface is connected to the angle computation unit (ACU). The ACU evaluates this validity information. Typically only valid encoder positions are used by the ACU. If the ACU receives invalid values, the ACU automatically switches to the sensor that provides valid values.</p> <p>DioCl2EncoderIn_setEncPosValidity is useful if the encoder does not provide an index signal.</p> <p>The setting does not take effect until you call DioCl2EncoderIn_apply during the initialization phase or DioCl2EncoderIn_write during run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.</p>								
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>								
Parameters	<p>pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using DioCl2EncoderIn_create.</p> <p>ValidityState Lets you specify the validity of the encoder position.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_ENC_POSITION_INVALID</td><td>Specifies the encoder position as invalid</td></tr> <tr> <td>DIO_ENC_POSITION_VALID</td><td>Specifies the encoder position as valid</td></tr> </tbody> </table>	Symbol	Meaning	DIO_ENC_POSITION_INVALID	Specifies the encoder position as invalid	DIO_ENC_POSITION_VALID	Specifies the encoder position as valid		
Symbol	Meaning								
DIO_ENC_POSITION_INVALID	Specifies the encoder position as invalid								
DIO_ENC_POSITION_VALID	Specifies the encoder position as valid								
Return value	The function returns an error code.								
Related topics	<p>References</p> <table border="1"> <tbody> <tr> <td>DioCl2EncoderIn_apply.....</td><td>637</td></tr> <tr> <td>DioCl2EncoderIn_setEncPosition.....</td><td>649</td></tr> <tr> <td>DioCl2EncoderIn_start.....</td><td>663</td></tr> <tr> <td>DioCl2EncoderIn_write.....</td><td>665</td></tr> </tbody> </table>	DioCl2EncoderIn_apply.....	637	DioCl2EncoderIn_setEncPosition.....	649	DioCl2EncoderIn_start.....	663	DioCl2EncoderIn_write.....	665
DioCl2EncoderIn_apply.....	637								
DioCl2EncoderIn_setEncPosition.....	649								
DioCl2EncoderIn_start.....	663								
DioCl2EncoderIn_write.....	665								

DioCl2EncoderIn_setGatedMode

Syntax

```
UInt32 DioCl2EncoderIn_setGatedMode(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    UInt32 GatedMode)
```

Include file

`IoDrvDioClass2EncoderIn.h`

Purpose

To specify the gated mode of the incremental encoder's IDX signal.

Description

This function is relevant only if you want to use the encoder's index signal (IDX).

The function is used to specify whether the *IDX* is gated to the *PHI0* and *PHI90* signals. This means that, the *IDX* signal is ignored unless *PHI0* and *PHI90* are both *High*.

These settings do not take effect until you call **DioCl2EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using **DioCl2EncoderIn_create**.

GatedMode Lets you specify whether the IDX signal is gated.

Symbol	Meaning
DIO_ENC_GMODE_DISABLED	Disables the gated mode of the incremental encoder's IDX signal (default).
DIO_ENC_GMODE_ENABLED	Enables the gated mode of the incremental encoder's IDX signal.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_setIndexMode.....	653
DioCl2EncoderIn_setIndexPosition.....	654

DioCl2EncoderIn_setIndexMode

Syntax

```
UInt32 DioCl2EncoderIn_setIndexMode(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    UInt32 IndexMode)
```

Include file

IoDrvDioClass2EncoderIn.h

Purpose

To specify how to react to an encoder index signal.

Description

This function is relevant only if you want to use the encoder's index signal (IDX).

The function is used to reset the encoder's position counter to the specified index position either at each index pulse or only at the first index pulse.

The default index position is 0.00. You can change its value via **DioCl2EncoderIn_setIndexPosition**.

These settings do not take effect until you call **DioCl2EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using **DioCl2EncoderIn_create**.

IndexMode Lets you specify how to react to an IDX signal.

Symbol	Meaning
DIO_ENC_IMODE_DISABLED	Specifies not to use the IDX signal (default).
DIO_ENC_EVERY_INDEX	Specifies to reset the position counter at every IDX pulse.
DIO_ENC_FIRST_INDEX	Specifies to reset the position counter only at the first IDX pulse.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_setGatedMode.....	652
DioCl2EncoderIn_setIndexPosition.....	654

DioCl2EncoderIn_setIndexPosition

Syntax

```
UInt32 DioCl2EncoderIn_setIndexPosition(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    Float64 IndexPos)
```

Include file

IoDrvDioClass2EncoderIn.h

Purpose

To specify the position that is written to the position counter when an index signal pulse occurs.

Description

This function is relevant only if the following conditions are fulfilled:

- You specify to use the encoder's index signal (IDX) via **DioCl2EncoderIn_create**.
- You enable the IDX signal via **DioCl2EncoderIn_setIndexMode**.

The index position that you specify must be in the defined position range. You can change the position range via **DioCl2EncoderIn_setMinPosition** and **DioCl2EncoderIn_setMaxPosition**.

These settings do not take effect until you call **DioCl2EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using **DioCl2EncoderIn_create**.

IndexPos Lets you specify the index position in lines in the range -2,097,152.00 ... 2,097,151.75 with a resolution of 0.25 lines. The default index position is 0.00.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_setIndexMode.....	653
DioCl2EncoderIn_setMaxPosition.....	657
DioCl2EncoderIn_setMinPosition.....	661

[DioCl2EncoderIn_setInputFilter](#)**Syntax**

```
UInt32 DioCl2EncoderIn_setInputFilter(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    Float64 FilterInterval)
```

Include file	<code>IoDrvDioClass2EncoderIn.h</code>						
Purpose	To specify the time interval for the noise filtering of incremental encoder signals on DIO Class 2 input channels.						
Description	<p>Only those input signals will be processed which have constant values over a period of time longer than the FilterInterval parameter specifies.</p> <p>The input filter will be applied to all the channels that are controlled by the specified DioCl2EncoderIn driver object.</p> <p>These settings do not take effect until you call DioCl2EncoderIn_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using DioCl2EncoderIn_create.</p> <p>FilterInterval Lets you specify the minimum time interval for which the input signal(s) must provide a constant value that is to be used for processing. Shorter input signals will be filtered. You can specify filter intervals in seconds in the range 0 ... 0.01 s.</p> <p>The resolution/deviation depends on the specified range:</p> <ul style="list-style-type: none"> ▪ 0 ... 640 ns Resolution of 10 ns ▪ >640 ns ... 0.01 s Deviation of <10% 						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics

References

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638

DioCl2EncoderIn_setMaxPosition

Syntax

```
UInt32 DioCl2EncoderIn_setMaxPosition(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    Float64 MaximumPos)
```

Include file

IoDrvDioClass2EncoderIn.h

Purpose

To specify the encoder's maximum position value.

Description

This function is used to specify the maximum position value in lines for the encoder's position counter.

If the encoder counts forwards and its value reaches the maximum position value, the position counter is set to the minimum position value that you specified via **DioCl2EncoderIn_setMinPosition**.

If the encoder counts backwards and its value reaches the minimum position value, the position counter is set to the maximum position value.

These settings do not take effect until you call **DioCl2EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using **DioCl2EncoderIn_create**.

MaximumPos Lets you specify the maximum position in lines in the range -2,097,152.00 ... 2,097,151.75 lines with a resolution of 0.25 lines. The default value for the maximum position is +2,097,151.75 lines.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_getEncPosition.....	640
DioCl2EncoderIn_setMinPosition.....	661

DioCl2EncoderIn_setMeasurementInterval

Syntax

```
UInt32 DioCl2EncoderIn_SetMeasurementInterval(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    UInt32 MeasInterval)
```

Include file

IoDrvDioClass2EncoderIn.h

Purpose

To specify the measurement interval used to calculate the encoder's velocity.

Description

This function is used to specify how many measured values of the encoder are used to calculate its velocity. For example, if you specify a measurement interval of three, the velocity calculation is based on the average of the encoder data captured by the last three calls of **DioCl2EncoderIn_read**.

These settings do not take effect until you call **DioCl2EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Incremental Encoder Interface](#) ([MicroLabBox Features](#)).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using [DioCl2EncoderIn_create](#).

MeasInterval Lets you specify the number of measurement intervals in the range 1 ... 20. The default value is 1.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_getEncVelocity.....	642
DioCl2EncoderIn_read.....	647

[DioCl2EncoderIn_setMinEncVelocity](#)**Syntax**

```
UInt32 DioCl2EncoderIn_setMinEncVelocity(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    Float64 MinEncVelocity)
```

Include file

`IoDrvDioClass2EncoderIn.h`

Purpose

To specify the encoder's minimum velocity in lines per second.

Description

This function is used to specify the minimum velocity to be measured. If the velocity falls below the specified minimum value, the measured velocity is set to 0, representing a standstill detection of the motor.

These settings do not take effect until you call **DioCl2EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using **DioCl2EncoderIn_create**.

MinEncVelocity Lets you specify the minimum encoder velocity in lines in the range 1.0 ... 1.0e+6 lines per second.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_setMinMechVelocity.....	660

DioCl2EncoderIn_setMinMechVelocity

Syntax

```
UInt32 DioCl2EncoderIn_setMinMechVelocity(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    Float64 MinMechVelocity)
```

Include file

`IoDrvDioClass2EncoderIn.h`

Purpose

To specify the encoder's minimum velocity in revolutions per minute (rpm).

Description

This function is used to specify the minimum velocity to be measured. If the velocity falls below the specified minimum value, the measured velocity is set to 0, representing a standstill detection of the motor.

These settings do not take effect until you call **DioCl2EncoderIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using **DioCl2EncoderIn_create**.

MinMechVelocity Lets you specify the minimum encoder velocity in revolutions per minute in the range 0.01 ... 10,000.0 rpm.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_setMinEncVelocity.....	659

DioCl2EncoderIn_setMinPosition

Syntax

```
UInt32 DioCl2EncoderIn_setMinPosition(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn,
    Float64 MinimumPos)
```

Include file	<code>IoDrvDioClass2EncoderIn.h</code>						
Purpose	To specify the encoder's minimum position value.						
Description	<p>This function is used to specify the minimum position value in lines for the encoder's position counter.</p> <p>If the encoder counts backwards and its value reaches the minimum position value, the position counter is set to the maximum position value that you specified via <code>DioCl2EncoderIn_setMaxPosition</code>.</p> <p>If the encoder counts forwards and its value reaches the maximum position value, the position counter is set to the minimum position value.</p> <p>These settings do not take effect until you call <code>DioCl2EncoderIn_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using <code>DioCl2EncoderIn_create</code>.</p> <p>MinimumPos Lets you specify the minimum position in lines in the range -2,097,152.00 ... 2,097,151.75 with a resolution of 0.25 lines. The default value for the minimum position is -2,097,152.00 lines.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	References						
	<table border="1"> <tr> <td>DioCl2EncoderIn_apply.....</td><td>637</td></tr> <tr> <td>DioCl2EncoderIn_create.....</td><td>638</td></tr> </table>	DioCl2EncoderIn_apply	637	DioCl2EncoderIn_create	638		
DioCl2EncoderIn_apply	637						
DioCl2EncoderIn_create	638						

DioCl2EncoderIn_getEncPosition.....	640
DioCl2EncoderIn_setMaxPosition.....	657

DioCl2EncoderIn_start

Syntax

```
UInt32 DioCl2EncoderIn_start(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn)
```

Include file

`IoDrvDioClass2EncoderIn.h`

Purpose

To start measuring incremental encoder signals via DIO Class 2 input channels.

Description

This function is used to start the specified DioCl2EncoderIn driver so that it will activate its digital input channels. The digital input channels are enabled to start capturing the incremental encoder input signals.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using [DioCl2EncoderIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References								
	<table> <tr> <td>DioCl2EncoderIn_apply.....</td> <td>637</td> </tr> <tr> <td>DioCl2EncoderIn_create.....</td> <td>638</td> </tr> <tr> <td>DioCl2EncoderIn_read.....</td> <td>647</td> </tr> <tr> <td>DioCl2EncoderIn_write.....</td> <td>665</td> </tr> </table>	DioCl2EncoderIn_apply.....	637	DioCl2EncoderIn_create.....	638	DioCl2EncoderIn_read.....	647	DioCl2EncoderIn_write.....	665
DioCl2EncoderIn_apply.....	637								
DioCl2EncoderIn_create.....	638								
DioCl2EncoderIn_read.....	647								
DioCl2EncoderIn_write.....	665								

DioCl2EncoderIn_stop

Syntax	<pre>UInt32 DioCl2EncoderIn_stop(DioCl2EncoderInSDrvObject *pDioCl2EncoderIn)</pre>
Include file	<code>IoDrvDioClass2EncoderIn.h</code>
Purpose	To disable the driver object to measure incremental encoder signals via DIO Class 2 input channels.
Description	<p>This function is used to stop the specified DioCl2EncoderIn driver so that it will deactivate its digital input channels. The digital input channels are disabled to stop capturing the incremental encoder input signals.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Incremental Encoder Interface (MicroLabBox Features).</p>
Parameters	<p>pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using DioCl2EncoderIn_create.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_read.....	647
DioCl2EncoderIn_write.....	665

DioCl2EncoderIn_write

Syntax

```
UInt32 DioCl2EncoderIn_write(
    DioCl2EncoderInSDrvObject *pDioCl2EncoderIn)
```

Include file

`IoDrvDioClass2EncoderIn.h`

Purpose

To update the settings of incremental encoder signal measurement via DIO Class 2 input channels during run time.

Description

If you have used the following function during run time, its settings will not be updated on the hardware until you call `DioCl2EncoderIn_write`:

- `DioCl2EncoderIn_setEncPosition`
- `DioCl2EncoderIn_setEncPosValidity`

The specified digital input channels must be enabled before by using `DioCl2EncoderIn_start`.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Incremental Encoder Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2EncoderIn Lets you specify the target incremental encoder via the corresponding DioCl2EncoderIn driver object. The DioCl2EncoderIn driver object had to be created before by using **DioCl2EncoderIn_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2EncoderIn_apply.....	637
DioCl2EncoderIn_create.....	638
DioCl2EncoderIn_setEncPosition.....	649
DioCl2EncoderIn_start.....	663

Resolver Interface

Introduction

You can access the resolver sensors via the `ResolverIn` functions.

Where to go from here

Information in this section

<code>ResolverIn_apply</code>	668
To apply the initialization settings to the resolver interface	
<code>ResolverIn_create</code>	669
To create the I/O driver object for operating a resolver sensor at the specified resolver interface.	
<code>ResolverIn_getFaultStatus</code>	671
To get the fault status information of the resolver interface.	
<code>ResolverIn_getIsDataValid</code>	673
To verify whether the provided electrical position and fault status information are valid.	
<code>ResolverIn_getPosition</code>	674
To get the current electrical resolver position.	
<code>ResolverIn_read</code>	675
To read data from a resolver interface.	
<code>ResolverIn_setExcitationFreq</code>	677
To specify the frequency of the excitation signal.	
<code>ResolverIn_setExcitationVoltRms</code>	678
To specify the voltage level of the excitation signal.	
<code>ResolverIn_setMaximumSpeed</code>	679
To specify the maximum rotational speed to be measured.	
<code>ResolverIn_setPositionOffset</code>	680
To specify an offset angle for the evaluated resolver position.	
<code>ResolverIn_setReverseDirection</code>	682
To specify that the resolver signals are interpreted as a reverse rotation.	
<code>ResolverIn_setSineCosineVoltRms</code>	683
To specify the voltage level of the input signals.	
<code>ResolverIn_start</code>	684
To start generating and measuring resolver sensor signals.	

Information in other sections

[Resolver Interface \(MicroLabBox Features\)](#) 

ResolverIn_apply

Syntax

```
UInt32 ResolverIn_apply(
    ResolverInSDrvObject *pResolverIn)
```

Include file

`IoDrvResolverIn.h`

Purpose

To apply the initialization settings to the resolver interface.

Description

Before you can start the specified resolver interface, you must transfer the settings of the `ResolverIn` driver object to its related input and output channels by using `ResolverIn_apply`.

You must do this also for the default values and for values that you specified via the `ResolverIn_set<Parameter>` functions. The `ResolverIn_apply` function is intended to be called at the end of the initialization phase of the real-time application.

The signal generation and measurement on the specified interface are not activated until you call `ResolverIn_start`.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters

pResolverIn Lets you specify the resolver interface to be accessed via the corresponding `ResolverIn` driver object. The `ResolverIn` driver object had to be created before by using `ResolverIn_create`.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics

References

ResolverIn_create.....	669
ResolverIn_start.....	684

ResolverIn_create

Syntax

```
UInt32 ResolverIn_create(
    ResolverInSDrvObject **ppResolverIn,
    UInt32 ResolverInUnitNo)
```

Include file

IoDrvResolverIn.h

Purpose

To create the I/O driver object for operating a resolver sensor at the specified resolver interface.

Description

This function is used to perform all the steps necessary to create an I/O driver object to access a resolver interface.

A ResolverIn driver object handles one resolver interface with output channels for generating the excitation voltage and input channels for measuring the induced voltages.

The resolver I/O driver is initialized with default values so that it is ready for use.

The initial values are:

- Excitation frequency: 10,000 Hz
- Excitation voltage: 7 V_{RMS}
- Sine/cosine input voltage: 3.5 V_{RMS}
- Maximum speed: 30,000 rpm (resolution of 14 bit)
- Rotational direction: Not inverted (according to the order of the resolver sensor signals)
- Offset added to resulting angle position: 0°

The function is intended to be called during the initialization phase of the real-time application.

You can use the **ResolverIn_set<Parameter>** functions to change the default parameters. Before you can start the specified resolver interface, you have to transfer the settings of the ResolverIn driver object to its related input and output channels by using **ResolverIn_apply**.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters

ppResolverIn Lets you specify the address of a variable which holds the address of the created driver object.

ResolverInUnitNo Lets you specify the resolver interface to be used for the created driver object in the range 1 ... 2. One driver object can handle one resolver interface. You can use the instance number to attach the sensor to an angle computation unit via [Acu_setInputSensor](#).

Symbol	Meaning
RESOLVER_INSTANCE_1	Specifies resolver interface 1.
RESOLVER_INSTANCE_2	Specifies resolver interface 2.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

ResolverIn_apply.....	668
ResolverIn_setExcitationFreq.....	677
ResolverIn_setExcitationVoltRms.....	678
ResolverIn_setMaximumSpeed.....	679
ResolverIn_setPositionOffset.....	680
ResolverIn_setReverseDirection.....	682
ResolverIn_setSineCosineVoltRms.....	683
ResolverIn_start.....	684

ResolverIn_getFaultStatus

Syntax

```
UInt32 ResolverIn_getFaultStatus(
    ResolverInSDrvObject *pResolverIn,
    UInt32 *pFaultStatus)
```

Include file

`IoDrvResolverIn.h`

Purpose

To get the fault status information of the resolver interface.

Description

This function is used to get information on the status of the resolver interface. The measured position might be valid only if no error is found.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call `ResolverIn_read` to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters

pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using `ResolverIn_create`.

pFaultStatus Lets you specify the address of a variable that holds the current fault status information of the resolver interface. The fault status information contains one or more of the following symbols combined via the bitwise OR operator.

Symbol	Meaning
<code>RESOLVER_CONFIG_PARITY_ERROR</code>	Indicates a <i>configuration parity</i> error. The register containing the configuration data of the resolver is corrupted.
<code>RESOLVER_PHASE_LOCK_ERROR</code>	Indicates a <i>phase lock</i> error. The difference between the phase of the excitation frequency and the phase of the sine and cosine signals exceeds the phase lock range, for example, caused by a <i>Velocity too high</i> error.

Symbol	Meaning
RESOLVER_VELOCITY_TOO_HIGH	Indicates a <i>velocity too high</i> error. The measured rotational speed exceeds the specified maximum speed. Specify a higher speed range by using ResolverIn_setMaximumSpeed .
RESOLVER_LOSS_OF_TRACKING	Indicates a <i>loss of tracking</i> error. The resolver interface cannot evaluate the position. The difference between the measured position and the internally calculated reference position of the resolver selftest exceeds the tolerance value. The tolerance value depends on the resolution of the speed range. 10 bit --> 12.5° max. tolerance 12 bit --> 5.0° max. tolerance 14 bit --> 2.5° max. tolerance 16 bit --> 2.5° max. tolerance
RESOLVER_DEGRAD_OF_SIG_MISMATCH	Indicates a <i>degradation of signal mismatch</i> error. The resolver selftest has detected that the amplitudes of the sine and cosine signals differ too much.
RESOLVER_DEGRAD_OF_SIG_OVERRANGE	Indicates a <i>degradation of signal overrange</i> error. The sine and cosine input signals exceed the tolerance values for the specified voltage level.
RESOLVER_INPUTS LOSS_OF_SIGNAL	Indicates an <i>inputs loss of signal</i> error. The captured values for the sine and cosine input signals are below the lower limit of the specified voltage level, for example, caused by an interrupted or disconnected signal.
RESOLVER_INPUTS CLIPPED	Indicates an <i>inputs clipped</i> error. The input signal of the sine or cosine signal exceeds the upper limit of the resolver interface. Before this error occurs, the protection circuit of the resolver interface will be activated.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

ResolverIn_read.....	675
ResolverIn_setMaximumSpeed.....	679
ResolverIn_start.....	684

ResolverIn_getIsDataValid

Syntax

```
UInt32 ResolverIn_getIsDataValid(
    ResolverInSDrvObject *pResolverIn,
    UInt32 *pIsDataValid)
```

Include file

`IoDrvResolverIn.h`

Purpose

To verify whether the provided electrical position and fault status information are valid.

Description

This function is used to evaluate whether the resolver interface is ready to receive data from the input signals. While the hardware is not able to get data from the input signals, the state of the interface is set to *invalid*. If the first data (position value and the fault status) has been received, the state is set to *valid* and stays valid.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call `ResolverIn_read` to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters

pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using `ResolverIn_create`.

pIsDataValid Lets you specify the address of a variable that holds the result of the validity check.

Value	Meaning
0	The resolver interface has not yet received data.
> 0	The resolver interface has received data, i.e., a position value and a fault status.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References						
	<table> <tr> <td>ResolverIn_read.....</td> <td>675</td> </tr> <tr> <td>ResolverIn_setMaximumSpeed.....</td> <td>679</td> </tr> <tr> <td>ResolverIn_start.....</td> <td>684</td> </tr> </table>	ResolverIn_read.....	675	ResolverIn_setMaximumSpeed.....	679	ResolverIn_start.....	684
ResolverIn_read.....	675						
ResolverIn_setMaximumSpeed.....	679						
ResolverIn_start.....	684						

ResolverIn_getPosition

Syntax	<pre>UInt32 ResolverIn_getPosition(ResolverInSDrvObject *pResolverIn, Float64 *pPosition)</pre>
Include file	IoDrvResolverIn.h
Purpose	To get the current electrical resolver position.
Description	<p>This function is used to provide the current electrical position that is evaluated by the resolver interface. The angle position value includes the optional offset specified by ResolverIn_setPositionOffset. Depending on the pole pairs of the resolver sensor and the pole pairs of the motor, the position value must be postprocessed to get the actual mechanical position of the motor.</p> <p>The resolution of the position is 0.055 degrees, which corresponds to a 16-bit resolution. Use the ResolverIn_getIsDataValid function to validate the measured position value.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call ResolverIn_read to update the internal buffer with the latest values.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Resolver Interface](#) ([MicroLabBox Features](#)).

Parameters	pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using ResolverIn_create .
	pPosition Lets you specify the address of a variable that holds the current electrical position of the resolver in degrees in the range 0° ... 359.9945° with a resolution of 0.0055°.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

ResolverIn_getFaultStatus.....	671
ResolverIn_getsDataValid.....	673
ResolverIn_read.....	675
ResolverIn_start.....	684

ResolverIn_read

Syntax

```
UInt32 ResolverIn_read(ResolverInSDrvObject *pResolverIn)
```

Include file

`IoDrvResolverIn.h`

Purpose

To read data from a resolver interface.

Description

This function is used to read the data from the resolver sensor connected to the resolver interface. It checks the resolver interface status information and calculates the electrical position. Then it evaluates the validity of the position and

the fault status and stores the results as a consistent data set in an internal buffer.

The input and output signals must be enabled before via **ResolverIn_start**.

To get the values from the internal buffer, you can use one of the following functions:

- **ResolverIn_getPosition** to get the value of the resolver position
- **ResolverIn_getIsDataValid** to get the information on whether the values of the electrical position and the fault status are valid
- **ResolverIn_getFaultStatus** to get the current fault information of the resolver interface

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters

pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using **ResolverIn_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

ResolverIn_getFaultStatus.....	671
ResolverIn_getIsDataValid.....	673
ResolverIn_getPosition.....	674
ResolverIn_start.....	684

ResolverIn_setExcitationFreq

Syntax

```
UInt32 ResolverIn_setExcitationFreq(
    ResolverInSDrvObject *pResolverIn,
    Float64 ExcitationFrequency)
```

Include file

`IoDrvResolverIn.h`

Purpose

To specify the frequency of the excitation signal.

Description

This function is used to specify the frequency used for generating the excitation output signal of the resolver interface.

The setting does not take effect until you call `ResolverIn_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters

pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using `ResolverIn_create`.

ExcitationFrequency Lets you specify the frequency of the excitation output signal in Hz in the range 2000 ... 20,000 Hz in steps of 250 Hz.

If the specified value is not an integer multiple of 250 Hz, it is automatically rounded up or down to the nearest integer multiple of 250. The value is rounded up if the difference of the available value and the specified value equals 125.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics**References**

ResolverIn_apply.....	668
ResolverIn_start.....	684

ResolverIn_setExcitationVoltRms

Syntax

```
UInt32 ResolverIn_setExcitationVoltRms(
    ResolverInSDrvObject *pResolverIn,
    UInt32 ExcitationVoltageSelector)
```

Include file

IoDrvResolverIn.h

Purpose

To specify the voltage level of the excitation signal.

Description

This function is used to specify the RMS (root mean square) voltage level used for generating the excitation output signal of the resolver interface. The required excitation signal voltage depends on the connected resolver.

The setting does not take effect until you call **ResolverIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters

pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using **ResolverIn_create**.

ExcitationVoltageSelector Lets you specify the voltage level of the excitation output signal in V_{RMS}.

Symbol	Meaning
RESOLVER_EXCITATION_3_0_V_RMS	Specifies a voltage level of 3.0 V _{RMS} for the excitation signal.
RESOLVER_EXCITATION_7_0_V_RMS	Specifies a voltage level of 7.0 V _{RMS} for the excitation signal.
RESOLVER_EXCITATION_10_0_V_RMS	Specifies a voltage level of 10.0 V _{RMS} for the excitation signal.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

ResolverIn_apply.....	668
ResolverIn_create.....	669
ResolverIn_setSineCosineVoltRms.....	683
ResolverIn_start.....	684

ResolverIn_setMaximumSpeed

Syntax

```
UInt32 ResolverIn_setMaximumSpeed(
    ResolverInSDrvObject *pResolverIn,
    UInt32 MaximumSpeedSelector)
```

Include file

IoDrvResolverIn.h

Purpose

To specify the maximum rotational speed to be measured.

Description

This function is used to specify the expected speed range of the motor in revolutions per minute. To get the highest possible resolution of the measurement, you should specify the lowest speed range applicable for your measurement.

The setting does not take effect until you call **ResolverIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters

pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before using `ResolverIn_create`.

MaximumSpeedSelector Lets you specify the maximum speed to be measured in revolutions per minute. By specifying the speed range, you choose the related resolution.

Symbol	Meaning
RESOLVER_MAX_SPEED_150000_RPM	Specifies a maximum speed of 150,000 rpm and a resolution of 10 bits.
RESOLVER_MAX_SPEED_60000_RPM	Specifies a maximum speed of 60,000 rpm and a resolution of 12 bits.
RESOLVER_MAX_SPEED_30000_RPM	Specifies a maximum speed of 30,000 rpm and a resolution of 14 bits.
RESOLVER_MAX_SPEED_7500_RPM	Specifies a maximum speed of 7500 rpm and a resolution of 16 bits.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

ResolverIn_apply.....	668
ResolverIn_create.....	669
ResolverIn_start.....	684

[ResolverIn_setPositionOffset](#)**Syntax**

```
UInt32 ResolverIn_setPositionOffset(
    ResolverInSDrvObject *pResolverIn,
    Float64 PositionOffset)
```

Include file	IoDrvResolverIn.h						
Purpose	To specify an offset angle for the evaluated resolver position.						
Description	<p>This function is used to specify an offset angle for the measured angular positions of the resolver sensor signals. For example, the offset value lets you trigger the commutation of the motor earlier (positive offset) or later (negative offset) than the measured resolver position.</p> <p>The setting does not take effect until you call ResolverIn_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Resolver Interface (MicroLabBox Features).</p>						
Parameters	<p>pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using ResolverIn_create.</p> <p>PositionOffset Lets you specify the position offset in degrees in the range -360.0° ... +360.0° with a resolution of 0.0055°.</p>						
Return value	The function returns an error code. <table border="1"><thead><tr><th>Error Code</th><th>Meaning</th></tr></thead><tbody><tr><td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr><tr><td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr></tbody></table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	References <table><tr><td>ResolverIn_apply.....</td><td>668</td></tr><tr><td>ResolverIn_start.....</td><td>684</td></tr></table>	ResolverIn_apply.....	668	ResolverIn_start.....	684		
ResolverIn_apply.....	668						
ResolverIn_start.....	684						

ResolverIn_setReverseDirection

Syntax	<pre>UInt32 ResolverIn_setReverseDirection(ResolverInSDrvObject *pResolverIn, UInt32 Reversing)</pre>
Include file	<code>IoDrvResolverIn.h</code>
Purpose	To specify that the resolver signals will be interpreted as a reverse rotation.
Description	<p>This function is useful if the resolver sensor has been installed in a wrong orientation or interchanged signals in a cable harness are to be compensated.</p> <p>The setting does not take effect until you call <code>ResolverIn_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Resolver Interface (MicroLabBox Features).</p>
Parameters	<p>pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using <code>ResolverIn_create</code>.</p> <p>Reversing Lets you specify whether the resolver signals will be interpreted as a reverse rotation.</p>
Symbol	Meaning
RESOLVER_REVERSE_INACTIVE	Specifies to interpret the resolver signals as a straight rotation according to the order of the resolver signals.
RESOLVER_REVERSE_ACTIVE	Specifies to interpret the resolver signals as a reverse rotation according to the order of the resolver signals.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

ResolverIn_apply.....	668
ResolverIn_create.....	669

ResolverIn_setSineCosineVoltRms

Syntax

```
UInt32 ResolverIn_setSineCosineVoltRms(
    ResolverInSDrvObject *pResolverIn,
    UInt32 SineCosineVoltageSelector)
```

Include file

IoDrvResolverIn.h

Purpose

To specify the voltage level of the input signals.

Description

This function is used to specify the RMS (root mean square) voltage level used for measuring the sine and cosine input signals of the resolver interface. The required induction signal voltage depends on the transformation ratio and the selected excitation signal voltage level of the connected resolver.

The setting does not take effect until you call **ResolverIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters	<p>pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using ResolverIn_create.</p> <p>SineCosineVoltageSelector Lets you specify the voltage level of the sine and cosine input signals in V_{RMS}.</p>
-------------------	---

Symbol	Meaning
RESOLVER_SINE_COSINE_1_5_V_RMS	Specifies a voltage level of 1.5 V _{RMS} for the sine and cosine signals.
RESOLVER_SINE_COSINE_3_5_V_RMS	Specifies a voltage level of 3.5 V _{RMS} for the sine and cosine signals.
RESOLVER_SINE_COSINE_5_0_V_RMS	Specifies a voltage level of 5.0 V _{RMS} for the sine and cosine signals.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References								
	<table border="0"> <tr> <td>ResolverIn_apply.....</td> <td>668</td> </tr> <tr> <td>ResolverIn_create.....</td> <td>669</td> </tr> <tr> <td>ResolverIn_setExcitationVoltRms.....</td> <td>678</td> </tr> <tr> <td>ResolverIn_start.....</td> <td>684</td> </tr> </table>	ResolverIn_apply.....	668	ResolverIn_create.....	669	ResolverIn_setExcitationVoltRms.....	678	ResolverIn_start.....	684
ResolverIn_apply.....	668								
ResolverIn_create.....	669								
ResolverIn_setExcitationVoltRms.....	678								
ResolverIn_start.....	684								

ResolverIn_start

Syntax	<pre>UInt32 ResolverIn_start(ResolverInSDrvObject *pResolverIn)</pre>
Include file	IoDrvResolverIn.h
Purpose	To start generating and measuring resolver sensor signals.
Description	This function is used to start the specified ResolverIn driver so that it will activate its analog input and output channels. The analog output channel is enabled to start generating the resolver excitation signal. The analog input channels are enabled to start capturing the resolver input signals.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Resolver Interface \(MicroLabBox Features\)](#).

Parameters

pResolverIn Lets you specify the resolver interface to be accessed via the corresponding ResolverIn driver object. The ResolverIn driver object had to be created before by using [ResolverIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

ResolverIn_apply.....	668
ResolverIn_create.....	669
ResolverIn_read.....	675

SSI Interface

Introduction

With the `DioCl2SsiIn` functions, you can access absolute encoders via the synchronous serial interface (SSI) on the DIO Class 2 channels to measure the position of a motor's rotor.

Where to go from here

Information in this section

<code>DioCl2SsiIn_apply</code>	687
To apply the initialization settings to the SSI interface.	
<code>DioCl2SsiIn_create</code>	689
To create the I/O driver object for operating an absolute encoder on the SSI interface via DIO Class 2 channels.	
<code>DioCl2SsiIn_getCommunicationErr</code>	691
To get information on communication errors between the SSI interface and the connected encoder.	
<code>DioCl2SsiIn_getIsPositionValid</code>	692
To get the result of the position validity check of the SSI interface.	
<code>DioCl2SsiIn_getIsRevolNoValid</code>	694
To check the validity of the revolution number provided by the SSI interface.	
<code>DioCl2SsiIn_getMechPosition</code>	695
To get the current mechanical position measured via the SSI interface.	
<code>DioCl2SsiIn_getRevolutionNo</code>	697
To get the current number of revolutions measured via the SSI interface.	
<code>DioCl2SsiIn_getSsiFrameData</code>	698
To get the complete bitstream of an SSI frame.	
<code>DioCl2SsiIn_getTransmitErrCount</code>	700
To get the total number of transmission errors.	
<code>DioCl2SsiIn_read</code>	701
To read data from the SSI interface.	
<code>DioCl2SsiIn_setClockFrequency</code>	703
To specify the frequency of the clock output signal of the SSI interface.	
<code>DioCl2SsiIn_setClockPause</code>	705
To specify the waiting time of the clock output signal of the SSI interface.	
<code>DioCl2SsiIn_setCodeType</code>	706
To specify the coding scheme of the absolute encoder connected via SSI interface.	

DioCl2SsiIn_setMultiturnRes	707
To specify the multi-turn resolution of an absolute encoder connected to the SSI interface.	
DioCl2SsiIn_setNoOfFirstPosBit	709
To specify the first bit in the transmission used for the position data.	
DioCl2SsiIn_setNumberOfSteps	710
To specify the number of steps of an absolute encoder connected to the SSI interface.	
DioCl2SsiIn_setPositionOffset	712
To specify an offset angle for the evaluated encoder position.	
DioCl2SsiIn_setRepeatReading	713
To specify whether to check the data consistency of the SSI interface.	
DioCl2SsiIn_setReverseDirection	715
To reverse the rotational direction in the SSI interface.	
DioCl2SsiIn_setSampleSyncEvSrc	716
To specify the event source triggering the data transmission to the SSI interface.	
DioCl2SsiIn_setSingleturnRes	718
To specify the single-turn resolution of an absolute encoder connected to the SSI interface.	
DioCl2SsiIn_setTransmitBitCount	719
To specify an encoder-specific number of bits per transmission.	
DioCl2SsiIn_setTransmitErrLimit	721
To specify the number of successive transmission errors to be ignored before a transmission error becomes effective.	
DioCl2SsiIn_start	722
To start generating and measuring signals via the SSI interface.	
DioCl2SsiIn_stop	723
To deactivate the SSI interface.	

DioCl2SsiIn_apply

Syntax

```
UInt32 DioCl2SsiIn_apply(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose	To apply the initialization settings to the SSI interface.						
Description	<p>Before you can start the specified SSI interface, you must transfer the settings of the DioCl2Ssiln driver object to its related input and output channels by using DioCl2SsiIn_apply.</p> <p>You must do this also for the default values and for values that you specified via the DioCl2SsiIn_set<Parameter> functions. The DioCl2SsiIn_apply function is intended to be called at the end of the initialization phase of the real-time application.</p> <p>The signal generation and measurement on the specified interface are not activated until you call DioCl2SsiIn_start.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>						
Parameters	pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using DioCl2SsiIn_create .						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	References						
	<table> <tr> <td>DioCl2Ssiln_create.....</td><td>689</td></tr> <tr> <td>DioCl2Ssiln_start.....</td><td>722</td></tr> </table>	DioCl2Ssiln_create	689	DioCl2Ssiln_start	722		
DioCl2Ssiln_create	689						
DioCl2Ssiln_start	722						

DioCl2SsiIn_create

Syntax

```
UInt32 DioCl2SsiIn_create(
    DioCl2SsiInSDrvObject **ppDioCl2SsiIn,
    UInt32 SsiInUnitNo,
    UInt32 Channel)
```

Include file

`IoDrvDicClass2SsiIn.h`

Purpose

To create the I/O driver object for operating an absolute encoder on the SSI interface via DIO Class 2 channels.

Description

This function is used to perform all the steps necessary to create an I/O driver object to access an SSI interface.

A DioCl2SsiIn driver object handles one SSI interface with one output channel for the clock signal (first channel) and one input channel for the data signal (second channel).

You allocate the required consecutive channels by specifying the first channel.

The I/O driver is initialized with default values so that it is ready for use.

The initial values are:

- Clock frequency: 1 MHz
- Encoder type: single-turn encoder
- Position offset: 0 degrees
- Reverse direction: do not reverse direction
- Single-turn resolution: 13 bit
- Multi-turn resolution: 0 bit
- Number of steps: DIO_CLASS2_SSI_DEFAULT_STEPS (corresponds to the single-turn resolution)
- Waiting time: 20 µs
- Coding scheme: Gray code
- Number of bits per transmission: 13 bit
- Starting bit of position data: 1 (first rising edge of the clock signal)
- Sample synchronization event: continuous
- Repeat reading: no (no data verification)
- Transmission error limit: 1

The function is intended to be called during the initialization phase of the real-time application.

You can use the `DioCl2SsiIn_set<Parameter>` functions to change the default parameters. Before you can start the specified SSI interface, you have to

transfer the settings of the DioCl2Ssiln driver object to its related I/O channels by using **DioCl2SsiIn_apply**.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

ppDioCl2Ssiln Lets you specify the address of a variable that holds the address of the created driver object for the SSI interface.

SsiInUnitNo Lets you specify the instance number of the SSI interface unit in the range 1 ... 2.

Symbol	Meaning
DIO_CLASS2_SSI_IN_UNIT_1	Specifies SSI interface 1.
DIO_CLASS2_SSI_IN_UNIT_2	Specifies SSI interface 2.

Channel Lets you specify the channel to be used for the clock output signal in the range 1 ... 11. The subsequent channel is used as the data channel.

These channels must not be allocated by other DIO Class 2 functions.

Symbol	Meaning
DIO_CLASS2_CHANNEL_1	Specifies channel 1 of DIO Class 2.
...	...
DIO_CLASS2_CHANNEL_11	Specifies channel 11 of DIO Class 2.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

[DioCl2Ssiln_apply](#)..... 687

DioCl2SsiIn_getCommunicationErr

Syntax

```
UInt32 DioCl2SsiIn_getCommunicationErr(  
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,  
    UInt32 *pCommunicationError)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To get information on communication errors between the SSI interface and the connected encoder.

Description

This function lets you get information about communication errors of the following two types:

- Connection error

A connection error is returned if no encoder is connected to the SSI interface, or a connected encoder does not respond.

- Transmission error

A transmission error is returned if the activated data verification returns a data inconsistency.

For further information on these error types and possible reasons, refer to [SSI Interface \(MicroLabBox Features\)](#).

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl2SsiIn_read** to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2SsiIn Lets you specify the DioCl2SsiIn driver object to be used for operating absolute encoders on the SSI interface. The DioCl2SsiIn object must already have been created by using **DioCl2SsiIn_create**.

pCommunicationError Lets you specify the address of a variable that holds the communication error information.

The following predefined symbols are used. They can be combined via the logical OR operator.

Symbol	Meaning
DIO_CLASS2_SSI_TRANSMIT_ERROR (0x01)	Indicates that the transmitted data is corrupt. The verification of the position data is executed only if you activated it before by using DioCl2SsiIn_setRepeatReading . Without data verification, no transmission error occurs. While the transmitted data is corrupt, the position data from the last valid transmission is returned.
DIO_CLASS2_SSI_NO_CONNECTION (0x02)	Indicates that no encoder is connected to the SSI interface, or that the connected encoder does not respond. The verification of the connection state is always activated.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2SsiIn_create.....	689
DioCl2SsiIn_read.....	701
DioCl2SsiIn_start.....	722

DioCl2SsiIn_getIsPositionValid

Syntax

```
UInt32 DioCl2SsiIn_getIsPositionValid(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt32 *pIsPositionValid)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To get the result of the position validity check of the SSI interface.

Description

This function is used to indicate whether the measured position that you read by using **DioCl2SsiIn_getMechPosition** is valid.

The validity check includes the following information:

- The consistency of the received position data if you activated reading the position data twice from the connected encoder using `DioCl2SsiIn_setRepeatReading`.
- The status of the connection to the absolute encoder, such as an exceeded waiting time.

A connection or transmission error, that represents a data inconsistency always results in an invalid position value. For transmission errors, you can specify a tolerance by using `DioCl2SsiIn_setTransmitErrLimit`.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call `DioCl2SsiIn_read` to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2SsiIn Lets you specify the DioCl2SsiIn driver object to be used for operating absolute encoders on the SSI interface. The DioCl2SsiIn object must already have been created by using `DioCl2SsiIn_create`.

plsPositionValid Lets you specify the address of a variable that holds the result of the position validity check.

Result Value	Meaning
0	The measured position value is not valid.
>0	The measured position value is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References										
	<table border="0"> <tr> <td>DioCl2SsiIn_create.....</td> <td>689</td> </tr> <tr> <td>DioCl2SsiIn_getMechPosition.....</td> <td>695</td> </tr> <tr> <td>DioCl2SsiIn_read.....</td> <td>701</td> </tr> <tr> <td>DioCl2SsiIn_setRepeatReading.....</td> <td>713</td> </tr> <tr> <td>DioCl2SsiIn_setTransmitErrLimit.....</td> <td>721</td> </tr> </table>	DioCl2SsiIn_create.....	689	DioCl2SsiIn_getMechPosition.....	695	DioCl2SsiIn_read.....	701	DioCl2SsiIn_setRepeatReading.....	713	DioCl2SsiIn_setTransmitErrLimit.....	721
DioCl2SsiIn_create.....	689										
DioCl2SsiIn_getMechPosition.....	695										
DioCl2SsiIn_read.....	701										
DioCl2SsiIn_setRepeatReading.....	713										
DioCl2SsiIn_setTransmitErrLimit.....	721										

DioCl2SsiIn_getIsRevoltNoValid

Syntax	<pre>UInt32 DioCl2SsiIn_getIsRevoltNoValid(DioCl2SsiInSDrvObject *pDioCl2SsiIn, UInt32 *pIsRevoltNoValid)</pre>
Include file	IoDrvDioClass2SsiIn.h
Purpose	To check the validity of the revolution number provided by the SSI interface.
Description	<p>This function is used to indicate whether the measured revolution number that you read by using DioCl2SsiIn_getRevolutionNo is valid.</p> <p>The validity check includes the following information:</p> <ul style="list-style-type: none"> ▪ The consistency of the received position data if you activated reading the position data twice from the connected encoder using DioCl2SsiIn_setRepeatReading. ▪ The status of the connection to the absolute encoder, such as an exceeded waiting time. ▪ The type of the connected encoder (single-turn or multi-turn encoder). <p>If a single-turn encoder is connected, a measured revolution number is always invalid. Only multi-turn encoders count the revolutions.</p> <p>A connection or transmission error, that represents a data inconsistency always results in an invalid revolution number. For transmission errors, you can specify a tolerance by using DioCl2SsiIn_setTransmitErrLimit.</p> <p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call DioCl2SsiIn_read to update the internal buffer with the latest values.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using [DioCl2SsiIn_create](#).

plsRevolutionNoValid Lets you specify the address of a variable that holds the result of the revolution number validity check.

Result Value	Meaning
0	The measured revolution number is not valid.
>0	The measured revolution number is valid.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2Ssiln_create.....	689
DioCl2Ssiln_getRevolutionNo.....	697
DioCl2Ssiln_read.....	701
DioCl2Ssiln_setRepeatReading.....	713
DioCl2Ssiln_setTransmitErrLimit.....	721

DioCl2Ssiln_getMechPosition

Syntax

```
UInt32 DioCl2SsiIn_getMechPosition(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    Float64 *pMechPosition)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose	To get the current mechanical position measured via the SSI interface.
Description	<p>This function is used to provide the current mechanical position that is evaluated by the SSI interface. The angle position value includes the optional offset specified by <code>DioCl2SsiIn_setPositionOffset</code>.</p> <p>The resolution of the position is $(360/\text{NumberOfSteps})$ degrees. The number of steps is specified by <code>DioCl2SsiIn_setNumberOfSteps</code>. Use <code>DioCl2SsiIn_getIsPositionValid</code> to validate the measured position value.</p>
<p>Note</p> <p>All channels of the DIO Class 2 unit are in a high impedance state after reset. To enable the clock output of the SSI interface that you specified by using <code>DioCl2SsiIn_create</code>, you must use <code>DioCl2SsiIn_start</code>.</p>	
<p>The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call <code>DioCl2SsiIn_read</code> to update the internal buffer with the latest values.</p>	
<p>The function is intended to be called during the run time of the real-time application.</p>	
<p>If an error is detected, the first instance of the error is returned to the caller.</p>	
<p>Note</p> <p>If you use an incremental encoder via the SSI interface, this function will not output the mechanical (absolute) angle position but the relative angle position determined by the incremental encoder. Usually, the starting point for the relative angle position is set when the incremental encoder is powered.</p>	

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) . For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features) .
Parameters	<p>pDioCl2SsiIn Lets you specify the DioCl2SsiIn driver object to be used for operating absolute encoders on the SSI interface. The DioCl2SsiIn object must already have been created by using <code>DioCl2SsiIn_create</code>.</p> <p>pMechPosition Lets you specify the address of a variable that holds the current mechanical position of the absolute encoder in degrees in the range $0^\circ \dots (360.0 - (360/\text{NumberOfSteps}))^\circ$ with a resolution of $(360/\text{NumberOfSteps})^\circ$.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2SsiIn_create.....	689
DioCl2SsiIn_getIsPositionValid.....	692
DioCl2SsiIn_read.....	701
DioCl2SsiIn_setNumberOfSteps.....	710
DioCl2SsiIn_setPositionOffset.....	712

DioCl2SsiIn_getRevolutionNo

Syntax

```
UInt32 DioCl2SsiIn_getRevolutionNo(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt32 *pRevolutionNo)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To get the current number of revolutions measured via the SSI interface.

Description

This function is used to provide the current number of revolutions, which is evaluated by the SSI interface. This value is available only when you use a multi-turn encoder. The revolution number starts with 0. The maximum number of revolutions depends on the multi-turn resolution that you specified by using [DioCl2SsiIn_setMultiturnRes](#).

Note

All channels of the DIO Class 2 unit are in a high impedance state after reset. To enable the clock output of the SSI interface that you specified by using [DioCl2SsiIn_create](#), you must use [DioCl2SsiIn_start](#).

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call [DioCl2SsiIn_read](#) to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using [DioCl2SsiIn_create](#).

pRevolutionNo Lets you specify the address of a variable that holds the number of the current revolution.

The maximum value is $2^{28}-1$.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2Ssiln_create.....	689
DioCl2Ssiln_getRevNoValid.....	694
DioCl2Ssiln_read.....	701
DioCl2Ssiln_setMultiturnRes.....	707
DioCl2Ssiln_start.....	722

DioCl2Ssiln_getSsiFrameData

Syntax

```
UInt32 DioCl2SsiIn_getSsiFrameData(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt8 *pSsiFrameData)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To get the complete bitstream of an SSI frame.

Description

Often, the transmitted serial bitstream provides only the revolution number and the position data. With the **DioCl2SsiIn_getRevolutionNo** and **DioCl2SsiIn_getMechPosition** functions, you can get these values even if you have to specify an offset to the first position bit by using **DioCl2SsiIn_setNoOfFirstPosBit**. If the encoder's bitstream provides additional bits before or after the position data and you want to evaluate them, you have to use **DioCl2SsiIn_getSsiFrame** to get the entire bitstream. Then, you have to evaluate the data by your own data processing method.

The data bits of the SSI frame are stored in the specified array in the following order:

- The most significant bit of the first element of the array (index 0) holds the most significant bit of the SSI frame transmitted by the encoder. Usually, this is the most significant bit of the position data when you use a single-turn encoder, or of the revolution data when you use a multi-turn encoder.
- The eighth bit of the SSI frame fills the least significant bit of the first array element.
- Unused bits in the last array element required to store the frame are filled with 0.
- Unused array elements remain in their initial states.

Note

The size of the array must be an integer multiple of 8 and at least the number of bits specified by **DioCl2SsiIn_setTransmitBitCount**.

The following example shows you an array providing 4 bytes to store an SSI frame of 13 bit.

Array	7						0	7					0	7				0	7				0
Frame data	0						1						2					3					0
Value	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	?	?	?	?	?	?	?	

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl2SsiIn_read** to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters	<p>pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using DioCl2SsiIn_create.</p> <p>pSsiFrameData Lets you specify the address of a variable that holds the data received from one SSI frame. The size of the frame depends on the specified SSI interface configuration.</p>
Return value	The function returns an error code.

Related topics		References								
		<table> <tr> <td>DioCl2Ssiln_create.....</td> <td>689</td> </tr> <tr> <td>DioCl2Ssiln_getCommunicationErr.....</td> <td>691</td> </tr> <tr> <td>DioCl2Ssiln_read.....</td> <td>701</td> </tr> <tr> <td>DioCl2Ssiln_setTransmitBitCount.....</td> <td>719</td> </tr> </table>	DioCl2Ssiln_create.....	689	DioCl2Ssiln_getCommunicationErr.....	691	DioCl2Ssiln_read.....	701	DioCl2Ssiln_setTransmitBitCount.....	719
DioCl2Ssiln_create.....	689									
DioCl2Ssiln_getCommunicationErr.....	691									
DioCl2Ssiln_read.....	701									
DioCl2Ssiln_setTransmitBitCount.....	719									

DioCl2Ssiln_getTransmitErrCount

Syntax	<pre>UInt32 DioCl2SsiIn_getTransmitErrCount(DioCl2SsiInSDrvObject *pDioCl2SsiIn, UInt32 *pTransmitErrorCount)</pre>
Include file	IoDrvDioClass2SsiIn.h
Purpose	To get the total number of transmission errors.
Description	If you activated the data verification using DioCl2SsiIn_setRepeatReading , each detected data inconsistency increments the transmission error counter provided by the SSI interface. The counter starts working when you load the real-time application and is able to count up to the maximum value of 65535.

The values are taken from the internal buffer of the I/O driver object. However, newer values might be available. Therefore, call **DioCl2SsiIn_read** to update the internal buffer with the latest values.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using **DioCl2SsiIn_create**.

pTransmitErrorCount Lets you specify the address of a variable that holds the total number of transmission errors in the range 0 ... 65535.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2Ssiln_create.....	689
DioCl2Ssiln_getCommunicationErr.....	691
DioCl2Ssiln_read.....	701

DioCl2Ssiln_read

Syntax

```
UInt32 DioCl2SsiIn_read(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose	To read data from the SSI interface.
Description	<p>This function is used to read the data from the absolute encoder connected to the SSI interface. It checks the SSI interface status (connection and transmission error information), evaluates the validity of the position and the revolution number and stores the results as a consistent data set in an internal buffer. The SSI interface is processing the received data according to the specified coding scheme used by the connected encoder and the direction of rotation.</p> <p>If you specified an offset by using <code>DioCl2SsiIn_setPositionOffset</code>, it is included in the read angle position.</p> <p>The input and output signals must have been enabled before via <code>DioCl2SsiIn_start</code>.</p> <p>To get the values from the internal buffer, you can use one of the following functions:</p> <ul style="list-style-type: none"> ▪ <code>DioCl2SsiIn_getMechPosition</code> to get the value of the mechanical position. ▪ <code>DioCl2SsiIn_getRevolutionNo</code> to get the number of the current revolution. ▪ <code>DioCl2SsiIn_getIsPositionValid</code> to get the information on whether the measured position value is valid. ▪ <code>DioCl2SsiIn_getIsRevolutionNoValid</code> to get the information on whether the measured revolution number is valid. ▪ <code>DioCl2SsiIn_getCommunicationErr</code> to get the information on communication errors, i.e., connection and transmission errors. ▪ <code>DioCl2SsiIn_getTransmitErrCount</code> to get the number of transmission errors. ▪ <code>DioCl2SsiIn_getSsiFrameData</code> to get the entire serial bitstream for processing additional information provided by the encoder. <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>
Parameters	<p>pDioCl2SsiIn Lets you specify the DioCl2SsiIn driver object to be used for operating absolute encoders on the SSI interface. The DioCl2SsiIn object must already have been created by using <code>DioCl2SsiIn_create</code>.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2Ssiln_create.....	689
DioCl2Ssiln_getCommunicationErr.....	691
DioCl2Ssiln_getIsPositionValid.....	692
DioCl2Ssiln_getIsRevoltNoValid.....	694
DioCl2Ssiln_getMechPosition.....	695
DioCl2Ssiln_getRevolutionNo.....	697
DioCl2Ssiln_getSsiFrameData.....	698
DioCl2Ssiln_getTransmitErrCount.....	700
DioCl2Ssiln_setCodeType.....	706
DioCl2Ssiln_setPositionOffset.....	712
DioCl2Ssiln_start.....	722

DioCl2Ssiln_setClockFrequency

Syntax

```
UInt32 DioCl2SsiIn_setClockFrequency(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    Float64 ClockFrequency)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To specify the frequency of the clock output signal of the SSI interface.

Description

Note the following restrictions on the configuration of the clock signal frequency:

- The SSI interface is able to generate only clock frequencies that are derived from 100 MHz divided by an integer value ($f_{\text{effective}} = 100 \text{ MHz} / n$).
- The effective clock frequency is calculated by using the two frequencies that result from the two neighboring integer values of the specified frequency. This gives you one frequency that is equal to or lower than the specified frequency, and one value that is equal to or higher than the specified one.
If the difference to the higher value exceeds 5%, the lower value is used.

If the difference to the higher value is smaller than or equal to 5%, the value closer to the specified value is used.

Example:

You have an absolute encoder that supports a clock frequency of 1.2 MHz.

```
fspecified = 1.2 MHz
n = 100 MHz / 1.2 MHz = 83.33
fhigher = 100 MHz / 83 = 1.2048 MHz
    (has a deviation of 0.4% that is <5%)
flower = 100 MHz / 84 = 1.19048 MHz
    (has a deviation of 0.8%)
```

In this example, the specified frequency results in an effective frequency of 1.2048 MHz.

- The maximum frequency that can be applied depends on the encoder and the length of the cable used to connect the encoder to the SSI interface. In general, the maximum cable length must not exceed 100 m.

The maximum clock frequency decreases as the cable length increases. For exact values, refer to the encoder's user documentation.

The setting takes effect only after you call **DioCl2SsiIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using **DioCl2SsiIn_create**.

ClockFrequency Lets you specify the frequency of the signal generated for the clock output of the SSI interface in Hz in the range 100 kHz ... 2 MHz.

Note that the maximum clock frequency depends on the cable length used.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2SsiIn_apply.....	687
DioCl2SsiIn_create.....	689
DioCl2SsiIn_setClockPause.....	705

DioCl2SsiIn_setClockPause

Syntax

```
UInt32 DioCl2SsiIn_setClockPause(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    Float64 ClockPause)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To specify the waiting time of the clock output signal of the SSI interface.

Description

There must be a pause in the clock signal between two transmissions. In this time, the encoder acquires new position data. Without a pause the encoder will output the previous data again.

Usually, the waiting time to be specified relates to the monoflop time stated in the encoder's user documentation. The monoflop time is the time interval between the last falling edge of the clock signal and the idle state of the data line. Refer to the encoder's user documentation to see whether your encoder requires a longer time interval than the monoflop time to be ready for new output.

If the specified waiting time does not match the encoder's requirements, a connection error occurs. For further information, refer to [DioCl2SsiIn_getCommunicationErr](#).

The setting takes effect only after you call [DioCl2SsiIn_apply](#) during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters	<p>pDioCl2SsiIn Lets you specify the DioCl2SsiIn driver object to be used for operating absolute encoders on the SSI interface. The DioCl2SsiIn object must already have been created by using DioCl2SsiIn_create.</p> <p>ClockPause Lets you specify the waiting time of the signal generated for the clock output of the SSI interface in seconds in the range 5.0e-6 ... 1.0 s in steps of 10 ns. For information on the minimum waiting time supported by your encoder (often specified as <i>monoflop time</i>), refer to the encoder's specification. If the specified value is not an integer multiple of 10 ns, the next higher value is used.</p>
Return value	The function returns an error code.

Related topics		References						
		<table> <tr> <td>DioCl2SsiIn_apply.....</td> <td>687</td> </tr> <tr> <td>DioCl2SsiIn_create.....</td> <td>689</td> </tr> <tr> <td>DioCl2SsiIn_setClockFrequency.....</td> <td>703</td> </tr> </table>	DioCl2SsiIn_apply.....	687	DioCl2SsiIn_create.....	689	DioCl2SsiIn_setClockFrequency.....	703
DioCl2SsiIn_apply.....	687							
DioCl2SsiIn_create.....	689							
DioCl2SsiIn_setClockFrequency.....	703							

DioCl2SsiIn_setCodeType

Syntax	<pre>UInt32 DioCl2SsiIn_setCodeType(DioCl2SsiInSDrvObject *pDioCl2SsiIn, UInt32 CodeType)</pre>
Include file	<code>IoDrvDioClass2SsiIn.h</code>
Purpose	To specify the coding scheme of the absolute encoder connected via SSI interface.
Description	<p>This function specifies the coding scheme with which the received data is to be interpreted.</p> <p>The setting takes effect only after you call DioCl2SsiIn_apply during the initialization phase.</p>

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using [DioCl2SsiIn_create](#).

CodeType Lets you specify the coding scheme of the revolution and position data supported by the connected encoder.

Coding Type	Meaning
DIO_CLASS2_SSI_CODE_GRAY	The revolution and position data received from your SSI encoder is interpreted as Gray coded.
DIO_CLASS2_SSI_CODE_BINARY	The revolution and position data received from your SSI encoder is interpreted as binary coded.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2Ssiln_apply	687
DioCl2Ssiln_create	689

DioCl2Ssiln_setMultiturnRes

Syntax

```
UInt32 DioCl2SsiIn_setMultiturnRes(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt32 MultiturnResolution)
```

Include file	<code>IoDrvDioClass2SsiIn.h</code>
Purpose	To specify the multi-turn resolution of an absolute encoder connected to the SSI interface.
Description	<p>The multi-turn resolution is a property of the connected encoder. It describes the number of bits that are used to provide the number of the current revolution. Implicitly, the multi-turn resolution defines the maximum number of revolutions the encoder is able to count.</p> <p>The configuration of the SSI interface depends on the type of the connected encoder:</p> <ul style="list-style-type: none"> ▪ Single-turn encoders You have to specify the single-turn resolution by using <code>DioCl2SsiIn_setSingleturnRes</code>. If the number of steps is not equal to $2^{\text{SingleturnResolution}}$, you have to explicitly specify the number of steps by using <code>DioCl2SsiIn_setNumberOfSteps</code>. You do not need to call the <code>DioCl2SsiIn_setMultiturnRes</code> function, but if you use it, you must specify the multi-turn resolution with 0. ▪ Multi-turn encoders You have to specify the single-turn resolution by using <code>DioCl2SsiIn_setSingleturnRes</code>, and the multi-turn resolution. The number of steps is then defined implicitly as $2^{\text{SingleturnResolution}}$. <p>If you specified resolution values that differ from the encoder specification, a temporary communication error (<code>DIO_CLASS2_SSI_NO_CONNECTION</code>) might occur. You can retrieve the error by using <code>DioCl2SsiIn_getCommunicationErr</code>.</p> <p>For further information on the encoder types, refer to SSI Interface (MicroLabBox Features).</p> <p>The setting takes effect only after you call <code>DioCl2SsiIn_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>
Parameters	<p>pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using <code>DioCl2SsiIn_create</code>.</p>

MultiturnResolution Lets you specify the number of bits that the connected encoder uses to provide the current number of revolutions in the range 0 ... 28 bit.

If you specify 0 bit, the SSI interface is configured for a single-turn encoder.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2SsiIn_apply.....	687
DioCl2SsiIn_create.....	689
DioCl2SsiIn_setNumberOfSteps.....	710
DioCl2SsiIn_setSingleturnRes.....	718

DioCl2SsiIn_setNoOfFirstPosBit

Syntax

```
UInt32 DioCl2SsiIn_setNoOfFirstPosBit(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt32 NumberOfFirstPositionBit)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To specify the first bit in the transmission used for the position data.

Description

This function has to be called if the encoder's transmission does not start transferring the position data at the first rising edge of the clock signal. If you use a single-turn encoder, the first bit of the position data is the first bit of the angular position value. If you use a multi-turn encoder, the first bit of the position data is the first bit of the revolution number. Usually, you have to configure the first bit of the position data if the encoder provides additional information during its transmission before the position data.

The setting takes effect only after you call **DioCl2SsiIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using [DioCl2SsiIn_create](#).

NumberOfFirstPositionBit Lets you specify the first bit of the position data in the transmission in the range 1 ... 64. The maximum value depends on the bitstream width and the encoder type.

```
Max = BitstreamWidth -  
      (ResolutionPerRevolution + RevolutionCounter) + 1
```

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2Ssiln_apply.....	687
DioCl2Ssiln_create.....	689
DioCl2Ssiln_setTransmitBitCount.....	719

DioCl2Ssiln_setNumberOfSteps

Syntax

```
UInt32 DioCl2SsiIn_setNumberOfSteps(  
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,  
    UInt32 NumberOfSteps)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose	To specify the number of steps of an absolute encoder connected to the SSI interface.
Description	<p>The number of steps is a property of the connected encoder. It describes into how many sections the connected encoder divides one revolution.</p> <p>The configuration of the SSI interface depends on the type of the connected encoder:</p> <ul style="list-style-type: none"> ▪ Single-turn encoders <p>If the connected encoder supports a number of steps smaller than $2^{\text{SingleTurnResolution}}$, you have to explicitly specify the number of steps. If the number of steps provided by the encoder corresponds to its single-turn resolution, you do not need to call this function. You can also use the DIO_CLASS2_SSI_DEFAULT_STEPS to make your source code more flexible.</p> <p>If you specified a value greater than $2^{\text{SingleTurnResolution}}$, the function aborts with an error message.</p> ▪ Multi-turn encoders <p>The number of steps is implicitly specified by the encoder's single-turn resolution. Calling DioCl2_SsiIn_setNumberOfSteps is not required. If you use this function, you have to specify $2^{\text{SingleTurnResolution}}$ or DIO_CLASS2_SSI_DEFAULT_STEPS for the number of steps. Otherwise, the function aborts with an error message.</p> <p>For further information on the encoder types, refer to SSI Interface (MicroLabBox Features).</p> <p>The setting takes effect only after you call DioCl2SsiIn_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>
Parameters	<p>pDioCl2SsiIn Lets you specify the DioCl2SsiIn driver object to be used for operating absolute encoders on the SSI interface. The DioCl2SsiIn object must already have been created by using DioCl2SsiIn_create.</p> <p>NumberOfSteps Lets you specify the number of steps provided by the connected encoder in the range 1 ... 2^{28}. With the DIO_CLASS2_SSI_DEFAULT_STEPS define, you can specify to use the number of steps implicitly specified by the encoder's single-turn resolution.</p>

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2SsiIn_apply.....	687
DioCl2SsiIn_create.....	689
DioCl2SsiIn_setSingleturnRes.....	718

DioCl2SsiIn_setPositionOffset

Syntax

```
UInt32 DioCl2SsiIn_setPositionOffset(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    Float64 PositionOffset)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To specify an offset angle for the evaluated encoder position.

Description

This function is used to specify an offset angle for the measured angular position of the absolute encoder. For example, the offset value lets you do the zero balance with the motor.

The setting takes effect only after you call **DioCl2SsiIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2SsiIn Lets you specify the DioCl2SsiIn driver object to be used for operating absolute encoders on the SSI interface. The DioCl2SsiIn object must already have been created by using **DioCl2SsiIn_create**.

PositionOffset Lets you specify the position offset in degrees in the range -360.0° ... +360.0° with a resolution of 0.0055°.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2SsiIn_apply.....	687
DioCl2SsiIn_create.....	689
DioCl2SsiIn_getMechPosition.....	695
DioCl2SsiIn_read.....	701

DioCl2SsiIn_setRepeatReading

Syntax

```
UInt32 DioCl2SsiIn_setRepeatReading(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt32 RepeatedReading)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To specify whether to check the data consistency of the SSI interface.

Description

Some absolute encoders provide additional information in the bitstream to verify the correctness of its contents, such as a parity bit or a checksum. If your encoder does not provide such features for verification, you can use the verification mechanism provided by the SSI interface. If you activate repeated reading, the position data is read twice from the sensor and then compared. If the results differ, a transmission error is detected. If the number of transmission errors exceeds the limit that you specified by using **DioCl2SsiIn_setTransmitErrLimit**, the measured position data is marked as invalid.

Note

- If you activated the verification of the position data, the effective bandwidth of the communication between the SSI interface and the connected encoder is reduced, because every read access is performed twice.
- The data verification only works correctly if the exact number of bits per transmission is specified. The number of bits used for reading without repeated reading might differ from the number of bits used for repeated reading. Refer to the encoder's user documentation for information on the bits used per transmission in both modes.
- The connected encoder must be able to transmit the same position data twice.

The setting takes effect only after you call `DioCl2SsiIn_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using `DioCl2SsiIn_create`.

RepeatedReading Lets you specify whether to check the position data.

Symbol	Meaning
DIO_CLASS2_SSI_REPEAT_INACTIVE	Specifies that the data is not verified by reading twice.
DIO_CLASS2_SSI_REPEAT_ACTIVE	Specifies that the data is verified by reading twice.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2SsiIn_apply.....	687
DioCl2SsiIn_create.....	689
DioCl2SsiIn_getCommunicationErr.....	691
DioCl2SsiIn_getIsPositionValid.....	692
DioCl2SsiIn_setTransmitBitCount.....	719

DioCl2SsiIn_setReverseDirection

Syntax

```
UInt32 DioCl2SsiIn_setReverseDirection(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt32 Reversing)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To reverse the rotational direction in the SSI interface.

Description

The rotational direction that is given by an absolute encoder can be reversed in the SSI interface that the encoder is connected to.

If there is no reversing, the forward rotation is usually assumed to be the clockwise rotation (from the front view of the motor shaft). For information on the default direction of the rotation, refer to the encoder's documentation. If you activated reversing, a clockwise rotation is measured as a backward rotation.

This function is useful if it was required to install the absolute encoder in inverse orientation.

The setting takes effect only after you call **DioCl2SsiIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

Note

All channels of the DIO Class 2 unit are in a high impedance state after reset. To enable the clock output of the SSI interface that you specified by using **DioCl2SsiIn_create**, you must use **DioCl2SsiIn_start**.

I/O mapping	For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration) .
	For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features) .

Parameters	<p>pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using DioCl2SsiIn_create.</p> <p>Reversing Lets you specify whether to reverse the encoder data in the SSI interface.</p>
-------------------	--

Symbol	Meaning
DIO_CLASS2_SSI_REVERSE_INACTIVE	Specifies to interpret increasing position values in the encoder data as forward rotation.
DIO_CLASS2_SSI_REVERSE_ACTIVE	Specifies to interpret increasing position values in the encoder data as backward rotation.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References						
	<table border="1"> <tbody> <tr> <td>DioCl2Ssiln_apply.....</td> <td>687</td> </tr> <tr> <td>DioCl2Ssiln_create.....</td> <td>689</td> </tr> <tr> <td>DioCl2Ssiln_start.....</td> <td>722</td> </tr> </tbody> </table>	DioCl2Ssiln_apply.....	687	DioCl2Ssiln_create.....	689	DioCl2Ssiln_start.....	722
DioCl2Ssiln_apply.....	687						
DioCl2Ssiln_create.....	689						
DioCl2Ssiln_start.....	722						

DioCl2Ssiln_setSampleSyncEvSrc

Syntax	<pre>UInt32 DioCl2SsiIn_setSampleSyncEvSrc(DioCl2SsiInSDrvObject *pDioCl2SsiIn, UInt32 SampleSyncEventSource)</pre>
---------------	--

Include file	<code>IoDrvDioClass2SsiIn.h</code>
---------------------	------------------------------------

Purpose	To specify the event source triggering the data transmission to the SSI interface.										
Description	<p>The SSI interface is able to synchronize the transmission of the position information with other events in the real-time application that are transferred via a trigger line. For example, you can use rising edges of a digital signal as a trigger event, or the detection of the middle of a period when using multichannel PWM signal generation.</p> <p>If you do not specify a trigger line number that the SSI interface is listening to, the data transmission is performed continuously.</p> <p>The setting takes effect only after you call DioCl2SsiIn_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>										
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>										
Parameters	<p>pDioCl2SsiIn Lets you specify the DioCl2SsiIn driver object to be used for operating absolute encoders on the SSI interface. The DioCl2SsiIn object must already have been created by using DioCl2SsiIn_create.</p> <p>SampleSyncEventSource Lets you specify the number of the trigger line used as the trigger source for data transmission via the SSI interface in the range 1 ... 16.. There is an additional symbol available to specify not to use a trigger line.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS2_TRIGGER_LINE_NONE</td><td>No trigger line is used. The data transmission is processed continuously.</td></tr> <tr> <td>DIO_CLASS2_TRIGGER_LINE_1</td><td>Specifies trigger line 1.</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td>DIO_CLASS2_TRIGGER_LINE_16</td><td>Specifies trigger line 16.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS2_TRIGGER_LINE_NONE	No trigger line is used. The data transmission is processed continuously.	DIO_CLASS2_TRIGGER_LINE_1	Specifies trigger line 1.	DIO_CLASS2_TRIGGER_LINE_16	Specifies trigger line 16.
Symbol	Meaning										
DIO_CLASS2_TRIGGER_LINE_NONE	No trigger line is used. The data transmission is processed continuously.										
DIO_CLASS2_TRIGGER_LINE_1	Specifies trigger line 1.										
...	...										
DIO_CLASS2_TRIGGER_LINE_16	Specifies trigger line 16.										
Return value	The function returns an error code.										
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.				
Error Code	Meaning										
IOLIB_NO_ERROR	The function was successfully completed.										
!= IOLIB_NO_ERROR	The function was not completed.										

Related topics**References**

DioCl2Ssiln_apply.....	687
DioCl2Ssiln_create.....	689

DioCl2Ssiln_setSingletturnRes

Syntax

```
UInt32 DioCl2SsiIn_setSingletturnRes(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt32 SingletturnResolution)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To specify the single-turn resolution of an absolute encoder connected to the SSI interface.

Description

The single-turn resolution is a property of the connected encoder. It describes the number of bits that are used to provide the value of a specific angular position within one revolution.

The configuration of the SSI interface depends on the type of the connected encoder:

- Single-turn encoders

If the number of steps is not equal to $2^{\text{SingletturnResolution}}$, you have to explicitly specify the number of steps by using **DioCl2SsiIn_setNumberOfSteps**. You do not need to call the **DioCl2SsiIn_setMultiturnRes** function, but if you use it, you must specify the multi-turn resolution with 0.

- Multi-turn encoders

You have to specify the single-turn resolution and the multi-turn resolution by using **DioCl2SsiIn_setMultiturnRes**. The number of steps is then defined implicitly as $2^{\text{SingletturnResolution}}$.

If you specified resolution values that differ from the encoder specification, a temporary communication error (**DIO_CLASS2_SSI_NO_CONNECTION**) might occur. You can retrieve the error by using

DioCl2SsiIn_getCommunicationErr.

For further information on the encoder types, refer to **SSI Interface (MicroLabBox Features)**.

The setting takes effect only after you call `DioCl2SsiIn_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using `DioCl2SsiIn_create`.

SingleturnResolution Lets you specify the number of bits that the connected encoder uses to provide an angular position within one revolution in the range 1 ... 28 bit.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2Ssiln_apply.....	687
DioCl2Ssiln_create.....	689
DioCl2Ssiln_setMultiturnRes.....	707
DioCl2Ssiln_setNumberOfSteps.....	710

DioCl2Ssiln_setTransmitBitCount

Syntax

```
UInt32 DioCl2SsiIn_setTransmitBitCount(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt32 NumberOfBitsPerTransmission)
```

Include file

`IoDrvDioClass2SsiIn.h`

Purpose	To specify an encoder-specific number of bits per transmission.						
Description	<p>This function has to be called if the encoder provides the position data and additional information, such as a parity bit or a CRC checksum. You have to explicitly specify the total number of transmitted bits to ensure that the SSI interface generates the required number of rising edges on the clock signal. If this function is not used, or <code>DIO_CLASS2_SSI_DEFAULT_BIT_COUNT</code> is specified for the <code>NumberOfBitsPerTransmission</code> parameter, the sum of the single-turn and multi-turn resolutions is used as the number of bits per transmission.</p> <p>The setting takes effect only after you call <code>DioCl2SsiIn_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl2SsiIn Lets you specify the DioCl2SsiIn driver object to be used for operating absolute encoders on the SSI interface. The DioCl2SsiIn object must already have been created by using <code>DioCl2SsiIn_create</code>.</p> <p>NumberOfBitsPerTransmission Lets you specify the total number of bits per transmission in the range 1 ... 64. The minimum number of bits to be specified is the sum of the single-turn and the multi-turn resolutions. You can specify the associated value by using the <code>DIO_CLASS2_SSI_DEFAULT_BIT_COUNT</code> define. If you specified a smaller number than the required minimum number of bits, <code>DioCl2SsiIn_apply</code> returns an error.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						

Related topics

References

DioCl2SsiIn_apply.....	687
DioCl2SsiIn_create.....	689
DioCl2SsiIn_setSingleturnRes.....	718

[DioCl2SsiIn_setTransmitErrLimit](#)

Syntax

```
UInt32 DioCl2SsiIn_setTransmitErrLimit(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn,
    UInt32 TransmitErrorLimit)
```

Include file

IoDrvDioClass2SsiIn.h

Purpose

To specify the number of successive transmission errors to be ignored before a transmission error becomes effective.

Description

If you activated the data verification by using **DioCl2SsiIn_setRepeatReading**, a data inconsistency results in a transmission error. Via the transmission error limit, you can specify the number of successive transmission errors that are to be ignored before the SSI interface generates a transmission error. For example, if you specify 1 as the error limit, there must be two successive errors to raise a transmission error.

If a transmission error occurs, the current values for the position and the revolution number are not valid, refer to **DioCl2SsiIn_getIsPositionValid** and **DioCl2SsiIn_getIsRevolutionValid**. The last valid data for position and revolution that was read by using **DioCl2SsiIn_getMechPosition** and **DioCl2SsiIn_getRevolutionNo**, is kept.

The setting takes effect only after you call **DioCl2SsiIn_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using [DioCl2SsiIn_create](#).

TransmitErrorLimit Lets you specify the number of successive transmission errors to be ignored before a transmission error becomes effective in the range 0 ... 255.

If you specify 0, the first transmission error becomes effective.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl2Ssiln_apply.....	687
DioCl2Ssiln_create.....	689
DioCl2Ssiln_getCommunicationErr.....	691
DioCl2Ssiln_getIsPositionValid.....	692
DioCl2Ssiln_getRevoltNoValid.....	694
DioCl2Ssiln_getTransmitErrCount.....	700
DioCl2Ssiln_setRepeatReading.....	713

DioCl2Ssiln_start

Syntax

```
UInt32 DioCl2SsiIn_start(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn)
```

Include file

`IoDrvDioClass2SsiIn.h`

Purpose

To start generating and measuring signals via the SSI interface.

Description

This function is used to start the specified SSI interface so that it activates its digital input and output channels. The first digital channel is enabled to start generating the Clock signal. The second digital channel is enabled to receive the Data signal for reading position values from the connected absolute encoder.

While the SSI interface is not started, the channels are in high impedance state.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [EnDat Interface \(MicroLabBox Features\)](#).

Parameters

pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using [DioCl2SsiIn_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl2Ssiln_apply.....	687
DioCl2Ssiln_create.....	689
DioCl2Ssiln_read.....	701
DioCl2Ssiln_stop.....	723

DioCl2Ssiln_stop

Syntax

```
UInt32 DioCl2SsiIn_stop(
    DioCl2SsiInSDrvObject *pDioCl2SsiIn)
```

Include file

`IoDrvDioClass2SsiIn.h`

Purpose

To deactivate the SSI interface.

Description	<p>This function is used to stop the specified SSI interface. Using DioCl2SsiIn_read in the stopped state leads to an error in the return code.</p> <p>When performing a stop/start transition for the SSI interface, the interface is not reset, e.g., the channels of the DIO Class 2 unit are not reset to the high impedance state and the transmission error counter continues with the last value when you restart the SSI interface.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to EnDat Interface (MicroLabBox Features).</p>						
Parameters	<p>pDioCl2Ssiln Lets you specify the DioCl2Ssiln driver object to be used for operating absolute encoders on the SSI interface. The DioCl2Ssiln object must already have been created by using DioCl2SsiIn_create.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"><thead><tr><th>Error Code</th><th>Meaning</th></tr></thead><tbody><tr><td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr><tr><td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr></tbody></table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics**References**

DioCl2Ssiln_create.....	689
DioCl2Ssiln_read.....	701
DioCl2Ssiln_start.....	722

Block-Commutated PWM Generation

Introduction

With the `DioCl1BcPwmOut` functions you can generate block-commutated pulse-width modulated signals on the DIO Class 1 channels to control an electric motor.

Where to go from here

Information in this section

DioCl1BcPwmOut_apply	727
To apply the initialization settings of the block-commutated signal generation to the DIO Class 1 output channels.	
DioCl1BcPwmOut_create	728
To create the I/O driver object for the block-commutated signal generation via DIO Class 1 output channels.	
DioCl1BcPwmOut_disableInterrupts	731
To disable the generation of interrupts on DIO Class 1 block-commutated PWM output channels.	
DioCl1BcPwmOut_enableInterrupts	732
To enable the generation of interrupts on DIO Class 1 block-commutated PWM output channels.	
DioCl1BcPwmOut_setAcu	734
To specify an angle computation unit (ACU) to provide the motor's rotor position.	
DioCl1BcPwmOut_setDirection	735
To set the rotational direction for the generation of block-commutated PWM signals at DIO Class 1 channels.	
DioCl1BcPwmOut_setDutyCycle	737
To set the duty cycle of the block-commutated PWM signals to be output at DIO Class 1 channels.	
DioCl1BcPwmOut_setEventDelay	738
To set the delay for events on DIO Class 1 output channels.	
DioCl1BcPwmOut_setEventDownsample	739
To set the downsampling for events on a DIO Class 1 output channel.	
DioCl1BcPwmOut_setInterruptMode	741
To specify the interrupt mode for the generation of block-commutated PWM output signals.	
DioCl1BcPwmOut_setIoIntVector	742
To set the interrupt handler for processing interrupts from block-commutated PWM signal generation on DIO Class 1 output channels.	
DioCl1BcPwmOut_setOutputHighZ	744
To set the high impedance state of the DIO Class 1 block-commutated PWM output channels.	

DioCl1BcPwmOut_setOutputMode	745
To select the output mode for the generation of block-commutated PWM signals at DIO Class 1 channels.	
DioCl1BcPwmOut_setPeriod	747
To set the period of the block-commutated PWM signals to be output at DIO Class 1 channels.	
DioCl1BcPwmOut_setPosition	748
To set the current motor position for the generation of block-commutated PWM signals at DIO Class 1 channels.	
DioCl1BcPwmOut_setPositionOffset	749
To specify an offset angle of the current rotor position for the block-commutated PWM signal generation	
DioCl1BcPwmOut_setRisingEdgeDelay	751
To specify a delay for rising edges of the generated PWM output signals.	
DioCl1BcPwmOut_setSectorCount	752
To specify the number of commutation sectors.	
DioCl1BcPwmOut_setSectorPositions	753
To specify the start positions of the commutation sectors.	
DioCl1BcPwmOut_setSectorSignals	754
To specify the sector-specific signal patterns for one output channel.	
DioCl1BcPwmOut_setSignalPairCheck	757
To enable signal pair check.	
DioCl1BcPwmOut_setSignalVoltage	758
To set the voltage level used for the DIO Class 1 block-commutated PWM output channels.	
DioCl1BcPwmOut_setStationarySignal	759
To generate a stationary signal at a DIO Class 1 channel used for block-commutated PWM signal generation.	
DioCl1BcPwmOut_setTriggerLineOut	761
To select the trigger line used for trigger signal generation.	
DioCl1BcPwmOut_setTriggerLineOutStatus	763
To set the status of the trigger line.	
DioCl1BcPwmOut_setTriggerMode	764
To specify the trigger mode for the generation of block-commutated PWM output signals.	
DioCl1BcPwmOut_setUpdateMode	766
To set the update mode of the DIO Class 1 block-commutated PWM output channels.	
DioCl1BcPwmOut_start	767
To start generating block-commutated signals via DIO Class 1 output channels.	

DioCl1BcPwmOut_stop..... 768

To stop generating block-commutated signals via DIO Class 1 output channels.

DioCl1BcPwmOut_write..... 769

To update the settings of the block-commutated signal generation for DIO Class 1 output channels during run time.

DioCl1BcPwmOut_apply

Syntax

```
UInt32 DioCl1BcPwmOut_apply(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To apply the initialization settings of the block-commutated signal generation to the DIO Class 1 output channels.

Description

Before you can start the block-commutated signal generation, you must transfer the settings of the DioCl1BcPwmOut driver object to the signal generation functionality by using **DioCl1BcPwmOut_apply**.

You must do this also for the default values and for values that you specified via the **DioCl1BcPwmOut_set<Parameter>** functions. The **DioCl1BcPwmOut_apply** function is intended to be called at the end of the initialization phase of the real-time application.

The signal generation on the specified digital output channels is not activated until you have called **DioCl1BcPwmOut_start**.

To update settings during run time, you must use **DioCl1BcPwmOut_write**.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters **pDioCl1BcPwmOut** Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using [DioCl1BcPwmOut_create](#).

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

Basics

[DioCl1BcPwmOut_setEventDownsample](#)..... 739

References

DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_setEventDelay.....	738
DioCl1BcPwmOut_setInterruptMode.....	741
DioCl1BcPwmOut_setIoIntVector.....	742
DioCl1BcPwmOut_setRisingEdgeDelay.....	751
DioCl1BcPwmOut_setSectorCount.....	752
DioCl1BcPwmOut_setSignalPairCheck.....	757
DioCl1BcPwmOut_setSignalVoltage.....	758
DioCl1BcPwmOut_setTriggerLineOut.....	761
DioCl1BcPwmOut_setTriggerMode.....	764
DioCl1BcPwmOut_setUpdateMode.....	766
DioCl1BcPwmOut_start.....	767

DioCl1BcPwmOut_create

Syntax

```
UInt32 DioCl1BcPwmOut_create(
    DioCl1BcPwmOutSDrvObject **ppDioCl1BcPwmOut,
    UInt32 BcPwmUnit,
    UInt32 Port,
    UInt32 FirstChannel,
    UInt32 ChannelCount)
```

Include file

[IoDrvDioClass1BcPwmOut.h](#)

Purpose

To create the I/O driver object for the block-commutated signal generation via DIO Class 1 output channels.

Description	<p>This function is used to perform all the steps necessary to create an I/O driver object to access DIO Class 1 channels.</p> <p>A DioCl1BcPwmOut driver object handles one block-commutated signal generation interface with output channels for the specified signals.</p> <p>You allocate the required consecutive channels by specifying the first channel.</p> <p>The I/O driver is initialized with default values so that it is ready for use.</p> <p>The initial values are:</p> <ul style="list-style-type: none"> ▪ ACU: No ACU selected ▪ Output signal voltage level: +2.5 V ▪ Update mode: Synchronization at the beginning of the next period ▪ Rising edge delay: 0 s ▪ PWM signal period: 1 ms ▪ PWM signal duty cycle: 0.5, i.e., 50%. ▪ Interrupt mode: No interrupt generation ▪ Trigger mode: No trigger signal generation ▪ Event downsample: 1 ▪ Rotational direction: Forward ▪ Event delay: 0 s ▪ Sector count: 6 ▪ Sector start positions: {0.0, 60.0, 120.0, 180.0, 240.0, 300.0} degrees ▪ Sector signals: All signals are constant <i>Low</i> ▪ Stationary signals: All signals are constant <i>Low</i> ▪ Output mode: Output stationary signals <p>The function is intended to be called during the initialization phase of the real-time application.</p> <p>You can use the <code>DioCl1BcPwmOut_set<Parameter></code> functions to change the default parameters. Before you can start the specified block-commutated signal generation, you have to transfer the settings of the DioCl1BcPwmOut driver object to its related output channels by using <code>DioCl1BcPwmOut_apply</code>.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Block-Commuted PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>
Parameters	<p>ppDioCl1BcPwmOut Lets you specify the address of a variable which holds the address of the created driver object for the block-commutated signal generation.</p>

BcPwmUnit Lets you specify the instance number of the block-commutated signal generation unit in the range 1 ... 2. One driver object can only handle one block-commutated signal generation interface.

Symbol	Meaning
DIO_BC_PWM_UNIT_1	Specifies signal generation interface 1.
DIO_BC_PWM_UNIT_2	Specifies signal generation interface 2.

Port Lets you specify the number of the DIO Class 1 port providing the required channels in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
...	...
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

FirstChannel Lets you specify the channel to be used for A+ of the signal generation in the range 1 ... 15. The consecutive channels are used in the following order: B+, C+, A-, B- and C-. These channels must not be allocated by other DIO Class 1 functions.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified DIO Class 1 port.
...	...
DIO_CLASS1_CHANNEL_15	Specifies channel 15 of the specified DIO Class 1 port.

ChannelCount Lets you specify the number of channels to be controlled by this driver in the range 2 ... 12. The channel count is not necessarily equal to the sector count.

To be able to check the signal pairs, the positive channels are counted first followed by the corresponding negative signal. For example, if the channel count is 6, the channels must be ordered as A+, B+, C+, A-, B-, and C-. By default, the signal pair check is enabled.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**Basics**

DioCl1BcPwmOut_setEventDownsample	739
---	-----

References

DioCl1BcPwmOut_apply	727
DioCl1BcPwmOut_setAcu	734
DioCl1BcPwmOut_setEventDelay	738
DioCl1BcPwmOut_setInterruptMode	741
DioCl1BcPwmOut_setIoIntVector	742
DioCl1BcPwmOut_setRisingEdgeDelay	751
DioCl1BcPwmOut_setSectorCount	752
DioCl1BcPwmOut_setSignalPairCheck	757
DioCl1BcPwmOut_setSignalVoltage	758
DioCl1BcPwmOut_setTriggerLineOut	761
DioCl1BcPwmOut_setTriggerMode	764
DioCl1BcPwmOut_setUpdateMode	766
DioCl1BcPwmOut_start	767

DioCl1BcPwmOut_disableInterrupts

Syntax

```
UInt32 DioCl1BcPwmOut_disableInterrupts(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 InterruptSelect)
```

Include file

IoDrvDioClass1BcPwmOut.h

Purpose

To disable the generation of interrupts on DIO Class 1 block-commutated PWM output channels.

Description

With this function, you can disable the generation of interrupts that you specified before by using [DioCl1BcPwmOut_setInterruptMode](#).

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using **DioCl1BcPwmOut_create**.

InterruptSelect Lets you select the interrupts that are to be disabled. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_BEGIN_PERIOD	Disables the generation of interrupts at the beginning of each period.
DIO_CLASS1_INT_MID_PERIOD	Disables the generation of interrupts at the middle of each period.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_enableInterrupts.....	732
DioCl1BcPwmOut_setInterruptMode.....	741
DioCl1BcPwmOut_setIoIntVector.....	742
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_write.....	769

DioCl1BcPwmOut_enableInterrupts

Syntax

```
UInt32 DioCl1BcPwmOut_enableInterrupts(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 InterruptSelect)
```

Include file

IoDrvDioClass1BcPwmOut.h

Purpose

To enable the generation of interrupts on DIO Class 1 block-commutated PWM output channels.

Description With this function, you can either enable the generation of interrupts or reenable the generation of interrupts that you disabled before by using [DioCl1BcPwmOut_disableInterrupts](#).

After calling [DioCl1BcPwmOut_start](#), interrupts are disabled.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters **pDioCl1BcPwmOut** Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using [DioCl1BcPwmOut_create](#).

InterruptSelect Lets you select the interrupts that are to be enabled. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_BEGIN_PERIOD	Enables the generation of interrupts at the beginning of each period.
DIO_CLASS1_INT_MID_PERIOD	Enables the generation of interrupts at the middle of each period.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_disableInterrupts.....	731
DioCl1BcPwmOut_setInterruptMode.....	741
DioCl1BcPwmOut_setIoIntVector.....	742
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_write.....	769

DioCl1BcPwmOut_setAcu

Syntax

```
UInt32 DioCl1BcPwmOut_setAcu(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 Acu)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To specify an angle computation unit (ACU) to provide the motor's rotor position.

Description

The current rotor position that controls which signals are generated can be provided by an ACU or by software.

The specified ACU must be created and started beforehand. For more details, refer to [Angle Computation Unit \(ACU\)](#) on page 493.

If you do not use an ACU for getting the motor position, you must continuously call [DioCl1BcPwmOut_setPosition](#).

The setting does not take effect until you call [DioCl1BcPwmOut_apply](#) during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using [DioCl1BcPwmOut_create](#).

Acu Lets you specify the angle computation unit to be used in the range 1 ... 4.

Symbol	Meaning
DIO_BC_PWM_ACU_INSTANCE_NONE	No ACU is used (default). The motor position must be set by software.
DIO_BC_PWM_ACU_INSTANCE_1	Specifies ACU number 1 to deliver the motor position.
...	...
DIO_BC_PWM_ACU_INSTANCE_4	Specifies ACU number 4 to deliver the motor position.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

Acu_create.....	496
DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728

DioCl1BcPwmOut_setDirection

Syntax

```
UInt32 DioCl1BcPwmOut_setDirection(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    Int32 Direction)
```

Include file

IoDrvDioClass1BcPwmOut.h

Purpose

To set the rotational direction for the generation of block-commutated PWM signals at DIO Class 1 channels.

Description

For a backward rotation, the block-commutated signal patterns are shifted. The output signals are generated for the sector that is 180° shifted to the nominal sector. The following table shows you an example for a block-commutated signal pattern with 6 sectors:

Nominal Sector	Sector of the Output Signal	
	Forward Rotation	Backward Rotation
1	1	4
2	2	5
3	3	6
4	4	1
5	5	2
6	6	3

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase or `DioCl1BcPwmOut_write` during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the block-commutated PWM output that you specified by using `DioCl1BcPwmOut_create`, you must use `DioCl1BcPwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

Direction Lets you specify the rotational direction.

Symbol	Meaning
DIO_BC_PWM_DIRECTION_FORWARD	Specifies to generate the signal patterns in nominal order (default).
DIO_BC_PWM_DIRECTION_BACKWARD	Specifies to generate the signal patterns in shifted order.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_setSectorPositions.....	753
DioCl1BcPwmOut_setSectorSignals.....	754
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_write.....	769

DioCl1BcPwmOut_setDutyCycle

Syntax

```
UInt32 DioCl1BcPwmOut_setDutyCycle(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    Float64 DutyCycle)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To set the duty cycle of the block-commutated PWM signals to be output at DIO Class 1 channels.

Description

The duty cycle is defined as the ratio between T_{High} and T_{Period} . You can specify an initial value for the duty cycle or you can update the duty cycle during run time of the real-time application.

The resolution of the duty cycle depends on the specified period value.

The resolution for time intervals within the PWM signal generator unit is 10 ns.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase or `DioCl1BcPwmOut_write` during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the block-commutated PWM output that you specified by using `DioCl1BcPwmOut_create`, you must use `DioCl1BcPwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

DutyCycle Lets you specify the duty cycle of the PWM signal to be generated in the range 0.0 ... 1.0, i.e., 0% ... 100%.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_setPeriod.....	747
DioCl1BcPwmOut_setSectorSignals.....	754
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_write.....	769

DioCl1BcPwmOut_setEventDelay

Syntax

```
UInt32 DioCl1BcPwmOut_setEventDelay(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    Float64 EventDelay)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To set the delay for events related to block-commutated signal generation.

Description

The function is used to delay the generation of events that you specified by using `DioCl1BcPwmOut_setInterruptMode` and `DioCl1BcPwmOut_setTriggerMode`.

The delay is the time interval between the occurrence of the event and the generation of the event signal (interrupt or trigger signal).

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#) ).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters	<p>pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using DioCl1BcPwmOut_create.</p> <p>EventDelay Lets you specify the time interval by which the generation of an event is delayed in seconds in the range 0 ... 1.34 s. You can specify the value in steps of 10 ns.</p>
Return value	The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	Basics								
	<table border="0"> <tr> <td style="background-color: #e0e0e0;">DioCl1BcPwmOut_setEventDownsample.....</td> <td style="background-color: #e0e0e0;">739</td> </tr> </table>	DioCl1BcPwmOut_setEventDownsample.....	739						
DioCl1BcPwmOut_setEventDownsample.....	739								
	References								
	<table border="0"> <tr> <td style="background-color: #e0e0e0;">DioCl1BcPwmOut_apply.....</td> <td style="background-color: #e0e0e0;">727</td> </tr> <tr> <td style="background-color: #e0e0e0;">DioCl1BcPwmOut_create.....</td> <td style="background-color: #e0e0e0;">728</td> </tr> <tr> <td style="background-color: #e0e0e0;">DioCl1BcPwmOut_setInterruptMode.....</td> <td style="background-color: #e0e0e0;">741</td> </tr> <tr> <td style="background-color: #e0e0e0;">DioCl1BcPwmOut_setTriggerMode.....</td> <td style="background-color: #e0e0e0;">764</td> </tr> </table>	DioCl1BcPwmOut_apply.....	727	DioCl1BcPwmOut_create.....	728	DioCl1BcPwmOut_setInterruptMode.....	741	DioCl1BcPwmOut_setTriggerMode.....	764
DioCl1BcPwmOut_apply.....	727								
DioCl1BcPwmOut_create.....	728								
DioCl1BcPwmOut_setInterruptMode.....	741								
DioCl1BcPwmOut_setTriggerMode.....	764								

DioCl1BcPwmOut_setEventDownsample

Syntax	<pre>UInt32 DioCl1BcPwmOut_setEventDownsample(DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut, UInt32 EventDownsampling)</pre>
Include file	<code>IoDrvDioClass1BcPwmOut.h</code>
Purpose	To set the downsampling for events related to block-commutated signal generation.

Description	<p>The function is used to reduce the number of generated events that you specified by using <code>DioCl1BcPwmOut_setInterruptMode</code> and <code>DioCl1BcPwmOut_setTriggerMode</code>.</p> <p>The setting does not take effect until you call <code>DioCl1BcPwmOut_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Block-Commutated PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using <code>DioCl1BcPwmOut_create</code>.</p> <p>EventDownsampling Lets you specify the downsampling value for event generation (interrupt or trigger signal) in the range 1 ... 256. The downsampling value specifies whether to generate an event on each occurrence of the event condition (EventDownsampling = 1, default) or only on each n-th occurrence, where n is the specified downsampling value.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics

References

<code>DioCl1BcPwmOut_apply</code>	727
<code>DioCl1BcPwmOut_create</code>	728
<code>DioCl1BcPwmOut_setEventDelay</code>	738

DioCl1BcPwmOut_setInterruptMode

Syntax

```
UInt32 DioCl1BcPwmOut_setInterruptMode(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 InterruptMode)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To specify the interrupt mode for the generation of block-commutated PWM output signals.

Description

This function is used to generate interrupts at certain points during the period of the output signals. You can select to generate interrupts at the period's beginning, its middle or both.

If you want to configure a combination of interrupts and trigger signals, you must configure the trigger mode in addition to the interrupt mode by using `DioCl1BcPwmOut_setTriggerMode`.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before using `DioCl1BcPwmOut_create`.

InterruptMode Lets you specify the interrupt mode for PWM signal generation. The following symbols can be combined by the bitwise OR operator.

Update Mode	Meaning
DIO_CLASS1_NO_INTERRUPT	No interrupt generation.
DIO_CLASS1_INT_BEGIN_PERIOD	Specifies to generate interrupts at the beginning of a period.
DIO_CLASS1_INT_MID_PERIOD	Specifies to generate interrupts at the middle of a period.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References										
	<table> <tbody> <tr> <td>DioCl1BcPwmOut_apply.....</td> <td>727</td> </tr> <tr> <td>DioCl1BcPwmOut_create.....</td> <td>728</td> </tr> <tr> <td>DioCl1BcPwmOut_disableInterrupts.....</td> <td>731</td> </tr> <tr> <td>DioCl1BcPwmOut_enableInterrupts.....</td> <td>732</td> </tr> <tr> <td>DioCl1BcPwmOut_setIoIntVector.....</td> <td>742</td> </tr> </tbody> </table>	DioCl1BcPwmOut_apply.....	727	DioCl1BcPwmOut_create.....	728	DioCl1BcPwmOut_disableInterrupts.....	731	DioCl1BcPwmOut_enableInterrupts.....	732	DioCl1BcPwmOut_setIoIntVector.....	742
DioCl1BcPwmOut_apply.....	727										
DioCl1BcPwmOut_create.....	728										
DioCl1BcPwmOut_disableInterrupts.....	731										
DioCl1BcPwmOut_enableInterrupts.....	732										
DioCl1BcPwmOut_setIoIntVector.....	742										

DioCl1BcPwmOut_setIoIntVector

Syntax	<pre>UInt32 DioCl1BcPwmOut_setIoIntVector(DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut, UInt32 InterruptSelect, IolibTToIntHandler InterruptHandler)</pre>
Include file	IoDrvDioClass1BcPwmOut.h
Purpose	To set the interrupt handler for processing interrupts from block-commutated PWM signal generation unit.
Description	<p>This function is used to register the function that handles the generated interrupts specified by using DioCl1BcPwmOut_setInterruptMode. For each specified condition for interrupt generation (beginning or middle of the period), you can define separate functions. If you want to handle all conditions with the same function, you can use the bitwise OR operator to combine the predefined symbols of the IntCondition parameter.</p> <p>Note</p> <p>The condition for interrupt generation must be a part of the conditions that are specified for the DioCl1BcPwmOut_setInterruptMode function.</p>

The setting does not take effect until you call **DioCl1BcPwmOut_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using [DioCl1BcPwmOut_create](#).

InterruptSelect Lets you select the condition the given interrupt handler is to be associated with. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_BEGIN_PERIOD	Specifies that the handler is to process interrupts generated at the beginning of the period.
DIO_CLASS1_INT_MID_PERIOD	Specifies that the handler is to process interrupts generated at the middle of the period.

InterruptHandler Lets you specify the address of the function which is to handle the generated interrupt.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_disableInterrupts.....	731
DioCl1BcPwmOut_enableInterrupts.....	732
DioCl1BcPwmOut_setInterruptMode.....	741

DioCl1BcPwmOut_setOutputHighZ

Syntax

```
UInt32 DioCl1BcPwmOut_setOutputHighZ(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 OutputHighZ)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To set the high impedance state of the DIO Class 1 block-commutated PWM output channels.

Description

You can force the output channels of the driver object to the high impedance state by setting the high impedance state to *On*. This is useful for termination purposes. If you want the real-time application to control the output levels, you can release the high impedance state again.

After calling `DioCl1BcPwmOut_start`, the high impedance state is *Off*.

The function is intended to be called during the run time of the real-time application.

The setting does not take effect until you call `DioCl1BcPwmOut_start` or `DioCl1BcPwmOut_write` during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the block-commutated PWM output that you specified by using `DioCl1BcPwmOut_create`, you must use `DioCl1BcPwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

OutputHighZ Lets you specify if outputs are to be set to the high impedance state or if outputs are to be released from the high impedance state.

Symbol	Meaning
DIO_CLASS1_HIGH_Z_OFF	Releases the high impedance state.
DIO_CLASS1_HIGH_Z_ON	Sets outputs to the high impedance state.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_setOutputMode.....	745
DioCl1BcPwmOut_setSignalVoltage.....	758
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_write.....	769

DioCl1BcPwmOut_setOutputMode

Syntax

```
UInt32 DioCl1BcPwmOut_setOutputMode(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 OutputMode)
```

Include file

IoDrvDioClass1BcPwmOut.h

Purpose

To select the output mode for the generation of block-commutated PWM signals at DIO Class 1 channels.

Description

The following output modes are available:

- The sector-dependent signal patterns that you specified via **DioCl1BcPwmOut_setSectorSignals** are generated.
- The stationary signals that you specified via **DioCl1BcPwmOut_setStationarySignal** are generated.

If no valid position is available, stationary signals are always output.

The setting does not take effect until you call `DioCl1BcPwmOut_write` during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the block-commutated PWM output that you specified by using `DioCl1BcPwmOut_create`, you must use `DioCl1BcPwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

OutputMode Lets you specify the output mode.

Symbol	Meaning
DIO_BC_PWM_OUTPUT_STATIONARY_SIGNALS	Specifies to generate the stationary signals (default).
DIO_BC_PWM_OUTPUT_SECTOR_SIGNALS	Specifies to generate the block-commutated PWM signals.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_setOutputHighZ.....	744
DioCl1BcPwmOut_setStationarySignal.....	759
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_write.....	769

DioCl1BcPwmOut_setPeriod

Syntax

```
UInt32 DioCl1BcPwmOut_setPeriod(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    Float64 Period)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To set the period of the block-commutated PWM signals to be output at DIO Class 1 channels.

Description

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase or `DioCl1BcPwmOut_write` during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the block-commutated PWM output that you specified by using `DioCl1BcPwmOut_create`, you must use `DioCl1BcPwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

Period Lets you specify the period of the PWM signals to be generated in seconds in the range 100 ns ... 1.34 s. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_setDutyCycle.....	737
DioCl1BcPwmOut_setSectorSignals.....	754
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_write.....	769

DioCl1BcPwmOut_setPosition

Syntax

```
UInt32 DioCl1BcPwmOut_setPosition(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    Float64 Position)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To set the current motor position for the generation of block-commutated PWM signals at DIO Class 1 channels.

Description

If you do not use an ACU to deliver the motor positions, you must continuously specify the positions directly in the I/O driver using this function.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase or `DioCl1BcPwmOut_write` during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the block-commutated PWM output that you specified by using `DioCl1BcPwmOut_create`, you must use `DioCl1BcPwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using [DioCl1BcPwmOut_create](#).

Position Lets you specify position of the rotor in degrees in the range 0° ... 360.0° with a resolution of 0.0055°.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_setAcu.....	734
DioCl1BcPwmOut_setPositionOffset.....	749
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_write.....	769

DioCl1BcPwmOut_setPositionOffset

Syntax

```
UInt32 DioCl1BcPwmOut_setPositionOffset(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    Float64 PositionOffset)
```

Include file

[IoDrvDioClass1BcPwmOut.h](#)

Purpose

To specify an offset angle of the current rotor position for the block-commutated PWM signal generation

Description

The offset angle is added to the current rotor position, no matter whether the position is set continuously via `DioCl1BcPwmOut_setPosition` or by using an angle computation unit (ACU) which you specify via `DioCl1BcPwmOut_setAcu`.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase or `DioCl1BcPwmOut_write` during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the block-commutated PWM output that you specified by using `DioCl1BcPwmOut_create`, you must use `DioCl1BcPwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

PositionOffset Lets you specify the offset angle for the position of the rotor in degrees in the range -60.0° ... +60.0° with a resolution of 0.0055°.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

<code>DioCl1BcPwmOut_apply</code>	727
<code>DioCl1BcPwmOut_setAcu</code>	734
<code>DioCl1BcPwmOut_setPosition</code>	748
<code>DioCl1BcPwmOut_start</code>	767
<code>DioCl1BcPwmOut_write</code>	769

DioCl1BcPwmOut_setRisingEdgeDelay

Syntax

```
UInt32 DioCl1BcPwmOut_setRisingEdgeDelay(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    Float64 RisingEdgeDelay)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To specify a delay for rising edges of the generated PWM output signals.

Description

The rising edges of the generated output signals will be output after a delay of the specified time. The falling edge is output without delay. This enables you to specify a dead time. Dead times are used, for example, to prevent shoot-through currents.

The function affects all output channels that are controlled by the specified driver object.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

RisingEdgeDelay Lets you specify the delay for rising edges in seconds in the range 0 ... 655 µs with a resolution of 10 ns. The default is 0 s.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728

DioCl1BcPwmOut_setSectorCount

Syntax

```
UInt32 DioCl1BcPwmOut_setSectorCount(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 SectorCount)
```

Include file

IoDrvDioClass1BcPwmOut.h

Purpose

To specify the number of commutation sectors.

Description

The function lets you specify the number of sectors for which you can configure an individual signal pattern via **DioCl1BcPwmOut_setSectorSignals**. The boundary of the sectors can be specified via **DioCl1BcPwmOut_setSectorPositions**.

The setting does not take effect until you call **DioCl1BcPwmOut_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using **DioCl1BcPwmOut_create**.

SectorCount Lets you specify the number of commutation sectors in the range 2 ... 12. The default is 6. The sector count is not necessarily equal to the channel count.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_setSectorPositions.....	753
DioCl1BcPwmOut_setSectorSignals.....	754

DioCl1BcPwmOut_setSectorPositions

Syntax

```
UInt32 DioCl1BcPwmOut_setSectorPositions(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 SectorPositionCount,
    const Float64 *pSectorPositions)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To specify the start positions of the commutation sectors.

Description

The function lets you specify the boundaries of the sectors for which you can configure an individual signal pattern via `DioCl1BcPwmOut_setSectorSignals`. The number of sectors can be specified via `DioCl1BcPwmOut_setSectorCount`.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters	<p>pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using DioCl1BcPwmOut_create.</p> <p>SectorPositionCount Lets you specify the length of the array that contains the start angles of the commutation sectors. The length must be the same as the parameter SectorCount of DioCl1BcPwmOut_setSectorCount.</p> <p>pSectorPositions Lets you specify the address of an array that stores the start position of each commutation sector in degrees in the range 0° ... 360.0° with a resolution of 0.0055°.</p> <p>For example, the array {0, 60, 120, 180, 240, 300} specifies the default values for six commutation sectors.</p>
Return value	The function returns an error code.

Related topics	References								
	<table> <tr> <td>DioCl1BcPwmOut_apply.....</td> <td>727</td> </tr> <tr> <td>DioCl1BcPwmOut_create.....</td> <td>728</td> </tr> <tr> <td>DioCl1BcPwmOut_setSectorCount.....</td> <td>752</td> </tr> <tr> <td>DioCl1BcPwmOut_setSectorSignals.....</td> <td>754</td> </tr> </table>	DioCl1BcPwmOut_apply.....	727	DioCl1BcPwmOut_create.....	728	DioCl1BcPwmOut_setSectorCount.....	752	DioCl1BcPwmOut_setSectorSignals.....	754
DioCl1BcPwmOut_apply.....	727								
DioCl1BcPwmOut_create.....	728								
DioCl1BcPwmOut_setSectorCount.....	752								
DioCl1BcPwmOut_setSectorSignals.....	754								

DioCl1BcPwmOut_setSectorSignals

Syntax	<pre>UInt32 DioCl1BcPwmOut_setSectorSignals(DioCl1BcPwmOutDrvObject *pDioCl1BcPwmOut, UInt32 ChannelId, UInt32 SectorSignalCount, const UInt32 *pSectorSignals)</pre>
Include file	<code>IoDrvDioClass1BcPwmOut.h</code>
Purpose	To specify the sector-specific signal patterns for one output channel.

Description

The function is used to specify a sequence of signal patterns that are generated for one DIO Class 1 output channel. The sequence represents a full rotation of the motor and consists of one signal pattern per commutation sector. You must call `DioCl1BcPwmOut_setSectorSignals` separately for each output channel.

The number of commutation sectors for one full rotation can be specified via `DioCl1BcPwmOut_setSectorCount`. The number of output channels is specified when you create the block-commutated signal generation via `DioCl1BcPwmOut_create`. The channel count is not necessarily equal to the sector count.

The period of the PWM signals can be specified via `DioCl1BcPwmOut_setPeriod`, the duty cycle via `DioCl1BcPwmOut_setDutyCycle`.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

ChannelId Lets you specify the channel to be used by an index number. For example, for a six-channel commutated motor control, the channels are specified in the following way:

Signal	ChannelId
A+	1
B+	2
C+	3
A-	4
B-	5
C-	6

Each channel identifier corresponds to one signal. Using the channel identifier makes your signal configuration independent from the channel number.

SectorSignalCount Lets you specify the length of the array that contains the output signals of the commutation sectors. This value must be equal to the parameter **SectorCount** of **DioCl1BcPwmOut_setSectorCount**.

pSectorSignals Lets you specify the address of an array that stores the signal pattern for each commutation sector.

Symbol	Meaning
DIO_BC_PWM_SIGNAL_CONST_LOW	Specifies to generate a constant <i>Low</i> output signal (default).
DIO_BC_PWM_SIGNAL_CONST_HIGH	Specifies to generate a constant <i>High</i> output signal.
DIO_BC_PWM_SIGNAL_PWM_NON_INVERTED	Specifies to generate a non-inverted PWM output signal.
DIO_BC_PWM_SIGNAL_PWM_INVERTED	Specifies to generate an inverted PWM output signal.
DIO_BC_PWM_SIGNAL_COMPLEMENT_PWM_NON_INVERTED	Specifies to generate a complemented, non-inverted PWM output signal. ¹⁾
DIO_BC_PWM_SIGNAL_COMPLEMENT_PWM_INVERTED	Specifies to generate a complemented, inverted PWM output signal. ²⁾

¹⁾ PWM signal specified by period and (1 - duty cycle) with a non-inverted output in the related sector set with **DioCl1BcPwmOut_setDutyCycle**

²⁾ PWM signal specified by period and (1 - duty cycle) with an inverted output in the related sector set with **DioCl1BcPwmOut_setDutyCycle**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_setSectorCount.....	752
DioCl1BcPwmOut_setSectorPositions.....	753

DioCl1BcPwmOut_setSignalPairCheck

Syntax

```
UInt32 DioCl1BcPwmOut_setSignalPairCheck(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 SignalPairCheck)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To set signal pair check.

Description

The function sets the validation of the block-commutated PWM output signals that you specified via `DioCl1BcPwmOut_setSectorSignals` and `DioCl1BcPwmOut_setStationarySignal`. This validation checks whether the output levels of each signal pair (e.g., A+ and A-) are both *High* at the same time. This is to avoid short circuits. At least one signal of each pair must always be *Low* or one signal must be the inverted of the other.

The check is applied when the real-time application is downloaded to the hardware. If the check is not passed, the real-time application does not start. By default, the check is enabled.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

SignalPairCheck Lets you enable the signal pair check.

Symbol	Meaning
IOLIB_TRUE	Enables signal pair check (default).
IOLIB_FALSE	Disables signal pair check.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728

DioCl1BcPwmOut_setSignalVoltage

Syntax

```
UInt32 DioCl1BcPwmOut_setSignalVoltage(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 SignalVoltage)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To set the voltage level used for the DIO Class 1 block-commutated PWM output channels.

Description

The output voltage level will be applied to the PWM output channels that are controlled by the specified DioCl1BcPwmOut driver object.

The setting does not take effect until you call **DioCl1BcPwmOut_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using [DioCl1BcPwmOut_create](#).

SignalVoltage Lets you select the voltage level used for the output signals.

Symbol	Meaning
DIO_CLASS1_SIGNAL_2_5_V	Specifies 2.5 V as the output voltage level (default).
DIO_CLASS1_SIGNAL_3_3_V	Specifies 3.3 V as the output voltage level.
DIO_CLASS1_SIGNAL_5_0_V	Specifies 5.0 V as the output voltage level.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_setOutputHighZ.....	744

DioCl1BcPwmOut_setStationarySignal

Syntax

```
UInt32 DioCl1BcPwmOut_setStationarySignal(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 ChannelId,
    UInt32 StationarySignal)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To generate a stationary signal at a DIO Class 1 channel used for block-commutated PWM signal generation.

Description	<p>This function lets you specify a stationary signal that is generated in the following cases:</p> <ul style="list-style-type: none"> ▪ No valid motor position is available. ▪ The stationary output mode is set via <code>DioCl1BcPwmOut_setOutputMode</code>. ▪ You want to apply a position-independent signal generation. <p>The stationary signal can be specified separately for each DIO Class 1 output channel.</p> <p>The setting does not take effect until you call <code>DioCl1BcPwmOut_apply</code> during the initialization phase or <code>DioCl1BcPwmOut_write</code> during run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.</p>
--------------------	--

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the block-commutated PWM output that you specified by using `DioCl1BcPwmOut_create`, you must use `DioCl1BcPwmOut_start`.

I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Block-Commuted PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>
--------------------	---

Parameters	<p>pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using <code>DioCl1BcPwmOut_create</code>.</p> <p>ChannelId Lets you specify the channel to be used by an index number. For example, for a six-channel commutated motor control, the channels are specified in the following way:</p>
-------------------	--

Signal	ChannelId
A+	1
B+	2
C+	3
A-	4
B-	5
C-	6

Each channel identifier corresponds to one signal. Using the channel identifier makes your signal configuration independent from the channel number.

StationarySignal Lets you specify the stationary signal to be generated at the specified channel.

Symbol	Meaning
DIO_BC_PWM_SIGNAL_CONST_HIGH	Specifies to generate a constant <i>High</i> signal.
DIO_BC_PWM_SIGNAL_CONST_LOW	Specifies to generate a constant <i>Low</i> signal.
DIO_BC_PWM_SIGNAL_PWM_NON_INVERTED	Specifies to generate a non-inverted PWM output signal.
DIO_BC_PWM_SIGNAL_PWM_INVERTED	Specifies to generate an inverted PWM output signal.
DIO_BC_PWM_SIGNAL_COMPLEMENT_PWM_NON_INVERTED	Specifies to generate a complemented, non-inverted PWM output signal. ¹⁾ .
DIO_BC_PWM_SIGNAL_COMPLEMENT_PWM_INVERTED	Specifies a complemented, inverted PWM output signal. ²⁾

¹⁾ PWM signal specified by period and (1 - duty cycle) with a non-inverted output in the related specific pattern set with `DioCl1BcPwmOut_setDutyCycle`

²⁾ PWM signal specified by period and (1 - duty cycle) with an inverted output in the related specific pattern with `DioCl1BcPwmOut_setDutyCycle`.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_setOutputHighZ.....	744
DioCl1BcPwmOut_setOutputMode.....	745
DioCl1BcPwmOut_start.....	767
DioCl1BcPwmOut_write.....	769

DioCl1BcPwmOut_setTriggerLineOut

Syntax

```
UInt32 DioCl1BcPwmOut_setTriggerLineOut(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 TriggerLineOut)
```

Include file	<code>IoDrvDioClass1BcPwmOut.h</code>								
Purpose	To select the trigger line used for trigger signal generation.								
Description	<p>A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board. You must specify the trigger line that the consumer listens to by using the relevant <code>xxx_setTriggerLineIn</code> function. A trigger line can be allocated only once.</p> <p>Before you can use trigger signals, you must set the trigger mode via <code>DioCl1BcPwmOut_setTriggerMode</code>. With <code>DioCl1BcPwmOut_setTriggerLineOutStatus</code> you enable or disable the trigger line.</p> <p>The setting does not take effect until you call <code>DioCl1BcPwmOut_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>								
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Block-Commutated PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>								
Parameters	<p>pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using <code>DioCl1BcPwmOut_create</code>.</p> <p>TriggerLineOut Lets you specify the number of the trigger line to be used for transmitting the trigger signal in the range 1 ... 16.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_TRIGGER_LINE_1</td><td>Specifies trigger line 1.</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td>DIO_CLASS1_TRIGGER_LINE_16</td><td>Specifies trigger line 16.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.	DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.
Symbol	Meaning								
DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.								
...	...								
DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.								
Return value	The function returns an error code.								
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.		
Error Code	Meaning								
IOLIB_NO_ERROR	The function was successfully completed.								
!= IOLIB_NO_ERROR	The function was not completed.								

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_setTriggerLineOutStatus.....	763
DioCl1BcPwmOut_setTriggerMode.....	764

DioCl1BcPwmOut_setTriggerLineOutStatus

Syntax

```
UInt32 DioCl1BcPwmOut_setTriggerLineOutStatus(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 TriggerLineStatus)
```

Include file

IoDrvDioClass1BcPwmOut.h

Purpose

To set the status of the trigger line.

Description

With this function, you can enable or disable the trigger line that you specified before by using **DioCl1BcPwmOut_setTriggerLineOut**.

The function is intended to be called during the initialization phase of the real-time application or during the run time of the application.

The setting does not take effect until you call **DioCl1BcPwmOut_start** during the initialization phase or during a run-to-stop or a stop-to-run transition of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using **DioCl1BcPwmOut_create**.

TriggerLineStatus Lets you specify the status of the trigger line.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_DISABLE	Disables the trigger line.
DIO_CLASS1_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_setTriggerLineOut.....	761
DioCl1BcPwmOut_start.....	767

DioCl1BcPwmOut_setTriggerMode

Syntax

```
UInt32 DioCl1BcPwmOut_setTriggerMode(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 TriggerMode)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To specify the trigger mode for the generation of block-commutated PWM output signals.

Description

This function is used to generate trigger signals related to the period of the output signals. You can select to generate trigger signals at the period's beginning, its middle or at both times.

Additionally, you must specify the trigger line for the generated trigger signal via `DioCl1BcPwmOut_setTriggerLineOut`. The I/O unit that is to consume the trigger signal has to specify this trigger line by using its related `xxx_setTriggerLineIn` function.

If you want to configure a combination of trigger signals and interrupts, you must configure the interrupt mode in addition to the trigger mode by using `DioCl1BcPwmOut_setInterruptMode`.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

TriggerMode Lets you specify the trigger mode for the PWM signal generation. The following symbols can be combined by the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_NO_TRIGGER	No trigger generation (default).
DIO_CLASS1_TRIG_BEGIN_PERIOD	Specifies to generate trigger events at the beginning of a period.
DIO_CLASS1_TRIG_MID_PERIOD	Specifies to generate trigger events at the middle of a period.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

<code>DioCl1BcPwmOut_apply</code>	727
<code>DioCl1BcPwmOut_create</code>	728
<code>DioCl1BcPwmOut_setInterruptMode</code>	741
<code>DioCl1BcPwmOut_setTriggerLineOut</code>	761

DioCl1BcPwmOut_setUpdateMode

Syntax

```
UInt32 DioCl1BcPwmOut_setUpdateMode(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut,
    UInt32 UpdateMode)
```

Include file

`IoDrvDioClass1BcPwmOut.h`

Purpose

To set the update mode of the DIO Class 1 block-commutated PWM output channels.

Description

The function is used to select the update behavior when you want to change the PWM period or duty cycle.

The setting does not take effect until you call `DioCl1BcPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using `DioCl1BcPwmOut_create`.

UpdateMode Lets you set the update mode to be used.

Symbol	Meaning
DIO_PWM_BEGIN_UPDATE	Specifies to update changes at the beginning of the next period (default).
DIO_PWM_MID_UPDATE	Specifies to update changes at the middle of the next period.
DIO_PWM_ANY_UPDATE	Specifies to update changes at the beginning and at the middle of the next period.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728

DioCl1BcPwmOut_start

Syntax

```
UInt32 DioCl1BcPwmOut_start(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut)
```

Include file

IoDrvDioClass1BcPwmOut.h

Purpose

To start generating block-commutated signals via DIO Class 1 output channels.

Description

This function is used to start the specified DioCl1BcPwmOut driver.

The driver activates the following points:

- It activates the block-commutated signal generation.
- It activates the digital output channels.

The digital output channels are enabled to generate the block-commutated PWM output signals. The output channels stay disabled if you call **DioCl1BcPwmOut_setOutputHighZ(DIO_CLASS1_HIGH_Z_ON)** before.

This function also enables the configured trigger line. If you call **DioCl1BcPwmOut_setTriggerLineOutStatus(DIO_CLASS1_TRIGGER_LINE_DISABLE)** before, the trigger line stays disabled.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commuted PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters	pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using DioCl1BcPwmOut_create .
-------------------	---

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics**References**

DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_setOutputHighZ.....	744
DioCl1EncoderIn_apply.....	606
DioCl1EncoderIn_write.....	634

DioCl1BcPwmOut_stop

Syntax

```
UInt32 DioCl1BcPwmOut_stop(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut)
```

Include file**IoDrvDioClass1BcPwmOut.h**

Purpose	To stop generating block-commutated signals via DIO Class 1 output channels.
----------------	--

Description

This function is used to stop the specified DioCl1BcPwmOut driver.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters	pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using DioCl1BcPwmOut_create .
-------------------	---

Return value	The function returns an error code.
---------------------	-------------------------------------

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply	727
DioCl1BcPwmOut_create	728
DioCl1BcPwmOut_write	769

DioCl1BcPwmOut_write

Syntax

```
UInt32 DioCl1BcPwmOut_write(
    DioCl1BcPwmOutSDrvObject *pDioCl1BcPwmOut)
```

Include file**IoDrvDioClass1BcPwmOut.h**

Purpose

To update the settings of the block-commutated signal generation for DIO Class 1 output channels during run time.

Description

If you have used one of the following functions during run time, their settings will not be updated on the hardware until you call [DioCl1BcPwmOut_write](#).

- [DioCl1BcPwmOut_setPeriod](#)
- [DioCl1BcPwmOut_setDutyCycle](#)
- [DioCl1BcPwmOut_setPosition](#)
- [DioCl1BcPwmOut_setPositionOffset](#)
- [DioCl1BcPwmOut_setDirection](#)
- [DioCl1BcPwmOut_setStationarySignal](#)
- [DioCl1BcPwmOut_setOutputMode](#)
- [DioCl1BcPwmOut_setOutputHighZ](#)

The specified digital output channels must be enabled before via [DioCl1BcPwmOut_start](#).

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Block-Commutated PWM Signal Generation \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1BcPwmOut Lets you specify the DioCl1BcPwmOut driver object to be used for block-commutated PWM signal generation. The DioCl1BcPwmOut object had to be created before by using [DioCl1BcPwmOut_create](#).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_create.....	728
DioCl1BcPwmOut_disableInterrupts.....	731
DioCl1BcPwmOut_enableInterrupts.....	732
DioCl1BcPwmOut_setDirection.....	735
DioCl1BcPwmOut_setDutyCycle.....	737
DioCl1BcPwmOut_setOutputHighZ.....	744
DioCl1BcPwmOut_setOutputMode.....	745
DioCl1BcPwmOut_setPeriod.....	747
DioCl1BcPwmOut_setPosition.....	748
DioCl1BcPwmOut_setPositionOffset.....	749
DioCl1BcPwmOut_setStationarySignal.....	759
DioCl1BcPwmOut_start.....	767

Multichannel PWM Signal Generation

Introduction

With the `DioCl1MultiPwmOut` functions, you can generate sine-commutated pulse-width modulated signals on the DIO Class 1 channels to control an electric motor.

Where to go from here

Information in this section

DioCl1MultiPwmOut_apply	772
To apply the initialization settings of the multichannel PWM signal generation to the DIO Class 1 output channels.	
DioCl1MultiPwmOut_create	774
To create the I/O driver object for the multichannel PWM signal generation via DIO Class 1 output channels.	
DioCl1MultiPwmOut_disableInterrupts	776
To disable the generation of interrupts on DIO Class 1 multichannel PWM output channels.	
DioCl1MultiPwmOut_enableInterrupts	777
To enable the generation of interrupts on DIO Class 1 multichannel PWM output channels.	
DioCl1MultiPwmOut_setAlignmentMode	779
To specify the alignment mode for the generation of multichannel PWM output signals.	
DioCl1MultiPwmOut_setDutyCycle	780
To set the duty cycles of the multichannel PWM signals to be output at DIO Class 1 channels.	
DioCl1MultiPwmOut_setEventDelay	781
To set the delay for events on DIO Class 1 output channels.	
DioCl1MultiPwmOut_setEventDownsample	783
To set the downsampling for events on a DIO Class 1 multichannel PWM output channel.	
DioCl1MultiPwmOut_setInterruptMode	784
To specify the interrupt mode for the generation of multichannel PWM output signals.	
DioCl1MultiPwmOut_setInvertingMode	785
To specify the inverting mode for the generation of multichannel PWM output signals.	
DioCl1MultiPwmOut_setIoIntVector	787
To set the interrupt handler for processing interrupts from multichannel PWM signal generation on DIO Class 1 output channels.	

DioCl1MultiPwmOut_setOutputHighZ	788
To set the high impedance state of the DIO Class 1 multichannel PWM output channels.	
DioCl1MultiPwmOut_setPeriod	790
To set the period of the multichannel PWM signals to be output at DIO Class 1 channels.	
DioCl1MultiPwmOut_setRisingEdgeDelay	791
To specify a delay for rising edges of the generated multichannel PWM output signals.	
DioCl1MultiPwmOut_setSignalVoltage	792
To specify the voltage level used for the DIO Class 1 multichannel PWM output channels.	
DioCl1MultiPwmOut_setTriggerLineOut	794
To select the trigger line used for trigger signal generation.	
DioCl1MultiPwmOut_setTriggerLineOutStatus	795
To set the status of the trigger line.	
DioCl1MultiPwmOut_setTriggerMode	796
To specify the trigger mode for the generation of multichannel PWM output signals.	
DioCl1MultiPwmOut_setUpdateMode	798
To set the update mode of the DIO Class 1 multichannel PWM output channels.	
DioCl1MultiPwmOut_start	799
To start generating multichannel PWM signals via DIO Class 1 output channels.	
DioCl1MultiPwmOut_write	801
To update the settings of the multichannel PWM signal generation for DIO Class 1 output channels during run time.	

DioCl1MultiPwmOut_apply

Syntax

```
UInt32 DioCl1MultiPwmOut_apply(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut)
```

Include file

`IoDrvDioClass1MultiPwmOut.h`

Purpose

To apply the initialization settings of the PWM multichannel signal generation to the DIO Class 1 output channels.

Description

Before you can start the specified DIO Class 1 channels, you must transfer the settings of the DioCl1MultiPwmOut driver object to its related output channels by using **DioCl1MultiPwmOut_apply**.

You must do this also for the default values and for values that you specified via the **DioCl1MultiPwmOut_set<Parameter>** functions. The **DioCl1MultiPwmOut_apply** function is intended to be called at the end of the initialization phase of the real-time application.

The signal generation on the specified digital output channels is not activated until you have called **DioCl1MultiPwmOut_start**.

To update settings during run time, you must use **DioCl1MultiPwmOut_write**.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using **DioCl1MultiPwmOut_create**.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1MultiPwmOut_create.....	774
DioCl1MultiPwmOut_setAlignmentMode.....	779
DioCl1MultiPwmOut_setEventDelay.....	781
DioCl1MultiPwmOut_setEventDownsample.....	783
DioCl1MultiPwmOut_setInterruptMode.....	784
DioCl1MultiPwmOut_setInvertingMode.....	785
DioCl1MultiPwmOut_setIoIntVector.....	787
DioCl1MultiPwmOut_setRisingEdgeDelay.....	791
DioCl1MultiPwmOut_setSignalVoltage.....	792
DioCl1MultiPwmOut_setTriggerLineOut.....	794
DioCl1MultiPwmOut_setTriggerMode.....	796

DioCl1MultiPwmOut_setUpdateMode.....	798
DioCl1MultiPwmOut_start.....	799

DioCl1MultiPwmOut_create

Syntax

```
UInt32 DioCl1MultiPwmOut_create(
    DioCl1MultiPwmOutSDrvObject **ppDioCl1MultiPwmOut,
    UInt32 Port,
    UInt32 Channel,
    UInt32 ChannelCount)
```

Include file

`IoDrvDioClass1MultiPwmOut.h`

Purpose

To create the I/O driver object for the multichannel PWM signal generation via DIO Class 1 output channels.

Description

This function is used to perform all the steps necessary to create an I/O driver object to access DIO Class 1 channels.

A DioCl1MultiPwmOut driver object handles one multichannel PWM interface.

You allocate the required consecutive channels by specifying the first channel.

The multichannel PWM signal driver is initialized with default values so that it is ready for use.

The initial values are:

- Output signal voltage level: +2.5 V
- Update mode: Synchronous update
- Alignment mode: Edge aligned
- Inverting mode: Do not generate inverted phase signal
- Rising edge delay: 0 s
- PWM signal period: 1 ms
- PWM signal duty cycle: 0.5, i.e., 50%.
- Interrupt mode: No interrupt generation
- Trigger mode: No trigger signal generation
- Event downsampling factor: 1
- Event delay: 0 s

The function is intended to be called during the initialization phase of the real-time application.

You can use the `DioCl1MultiPwmOut_set<Parameter>` functions to change the default parameters. Before you can start the specified DIO Class 1 channels, you have to transfer the settings of the DioCl1MultiPwmOut driver object to its related output channels by using `DioCl1MultiPwmOut_apply`.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

ppDioCl1MultiPwmOut Lets you specify the address of a variable which holds the address of the created driver object for the multichannel PWM signal generation.

Port Lets you specify the number of the DIO Class 1 port providing the required channels in the range 1 ... 3.

Symbol	Meaning
DIO_CLASS1_PORT_1	Specifies port 1 of the DIO Class 1 unit.
...	...
DIO_CLASS1_PORT_3	Specifies port 3 of the DIO Class 1 unit.

Channel Lets you specify the first channel to be used in the range 1 ... 16. The consecutive channels are used for the remaining channels to be controlled by this driver. These channels must not be allocated by other DIO Class 1 functions.

Symbol	Meaning
DIO_CLASS1_CHANNEL_1	Specifies channel 1 of the specified DIO Class 1 port.
...	...
DIO_CLASS1_CHANNEL_16	Specifies channel 16 of the specified DIO Class 1 port.

ChannelCount Lets you specify the number of channels to be controlled by this driver in the range 1 ... (17 - <Channel>).

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics	References
	<p>DioCl1MultiPwmOut_apply..... 772 DioCl1MultiPwmOut_setAlignmentMode..... 779 DioCl1MultiPwmOut_setEventDelay..... 781 DioCl1MultiPwmOut_setEventDownsample..... 783 DioCl1MultiPwmOut_setInterruptMode..... 784 DioCl1MultiPwmOut_setInvertingMode..... 785 DioCl1MultiPwmOut_setIoIntVector..... 787 DioCl1MultiPwmOut_setRisingEdgeDelay..... 791 DioCl1MultiPwmOut_setSignalVoltage..... 792 DioCl1MultiPwmOut_setTriggerLineOut..... 794 DioCl1MultiPwmOut_setTriggerMode..... 796 DioCl1MultiPwmOut_setUpdateMode..... 798 DioCl1MultiPwmOut_start..... 799</p>

DioCl1MultiPwmOut_disableInterrupts

Syntax	<pre>UInt32 DioCl1MultiPwmOut_disableInterrupts(DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut, UInt32 InterruptSelect)</pre>
Include file	IoDrvDioClass1MultiPwmOut.h
Purpose	To disable the generation of interrupts on DIO Class 1 multichannel PWM output channels.
Description	<p>With this function, you can disable the generation of interrupts that you specified before by using DioCl1BcPwmOut_setInterruptMode.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Multichannel PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>

Parameters	<p>pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using DioCl1MultiPwmOut_create.</p> <p>InterruptSelect Lets you select the interrupts that are to be disabled. The symbols can be combined by using the bitwise OR operator.</p>
-------------------	---

Symbol	Meaning
DIO_CLASS1_INT_BEGIN_PERIOD	Disables the generation of interrupts at the beginning of each period.
DIO_CLASS1_INT_MID_PERIOD	Disables the generation of interrupts at the middle of each period.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References												
	<table border="0"> <tr> <td>DioCl1MultiPwmOut_apply.....</td> <td>772</td> </tr> <tr> <td>DioCl1MultiPwmOut_enableInterrupts.....</td> <td>777</td> </tr> <tr> <td>DioCl1MultiPwmOut_setInterruptMode.....</td> <td>784</td> </tr> <tr> <td>DioCl1MultiPwmOut_setIntVector.....</td> <td>787</td> </tr> <tr> <td>DioCl1MultiPwmOut_start.....</td> <td>799</td> </tr> <tr> <td>DioCl1MultiPwmOut_write.....</td> <td>801</td> </tr> </table>	DioCl1MultiPwmOut_apply.....	772	DioCl1MultiPwmOut_enableInterrupts.....	777	DioCl1MultiPwmOut_setInterruptMode.....	784	DioCl1MultiPwmOut_setIntVector.....	787	DioCl1MultiPwmOut_start.....	799	DioCl1MultiPwmOut_write.....	801
DioCl1MultiPwmOut_apply.....	772												
DioCl1MultiPwmOut_enableInterrupts.....	777												
DioCl1MultiPwmOut_setInterruptMode.....	784												
DioCl1MultiPwmOut_setIntVector.....	787												
DioCl1MultiPwmOut_start.....	799												
DioCl1MultiPwmOut_write.....	801												

DioCl1MultiPwmOut_enableInterrupts

Syntax	<pre>UInt32 DioCl1MultiPwmOut_enableInterrupts(DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut, UInt32 InterruptSelect)</pre>
---------------	--

Include file	IoDrvDioClass1MultiPwmOut.h
---------------------	-----------------------------

Purpose	To enable the generation of interrupts on DIO Class 1 multichannel PWM output channels.
----------------	---

Description With this function, you can enable the generation of interrupts that you specified before by using **DioCl1MultiPwmOut_setInterruptMode**.

After calling **DioCl1MultiPwmOut_start**, interrupts are disabled.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters **pDioCl1MultiPwmOut** Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using **DioCl1MultiPwmOut_create**.

InterruptSelect Lets you select the interrupts that are to be enabled. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_BEGIN_PERIOD	Enables the generation of interrupts at the beginning of each period.
DIO_CLASS1_INT_MID_PERIOD	Enables the generation of interrupts at the middle of each period.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_disableInterrupts.....	776
DioCl1MultiPwmOut_setInterruptMode.....	784
DioCl1MultiPwmOut_setIoIntVector.....	787
DioCl1MultiPwmOut_start.....	799
DioCl1MultiPwmOut_write.....	801

DioCl1MultiPwmOut_setAlignmentMode

Syntax

```
UInt32 DioCl1MultiPwmOut_setAlignmentMode(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 AlignmentMode)
```

Include file

`IoDrvDioClass1MultiPwmOut.h`

Purpose

To specify the alignment mode for the generation of multichannel PWM output signals.

Description

The following alignment modes are available:

- The generated signals are aligned to their rising edges.
- The generated signals are aligned to the middle of their periods.

For more information on alignment modes, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

The setting does not take effect until you call `DioCl1MultiPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using `DioCl1MultiPwmOut_create`.

AlignmentMode Lets you specify the alignment mode for multichannel PWM signal generation.

Symbol	Meaning
<code>DIO_PWM_ALIGNMENT_EDGE</code>	Specifies to align the generated PWM signals to their rising edges.
<code>DIO_PWM_ALIGNMENT_CENTER</code>	Specifies to align the generated PWM signals to the middle of their periods.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References						
	<table> <tr> <td>DioCl1MultiPwmOut_apply.....</td> <td>772</td> </tr> <tr> <td>DioCl1MultiPwmOut_create.....</td> <td>774</td> </tr> <tr> <td>DioCl1MultiPwmOut_setUpdateMode.....</td> <td>798</td> </tr> </table>	DioCl1MultiPwmOut_apply.....	772	DioCl1MultiPwmOut_create.....	774	DioCl1MultiPwmOut_setUpdateMode.....	798
DioCl1MultiPwmOut_apply.....	772						
DioCl1MultiPwmOut_create.....	774						
DioCl1MultiPwmOut_setUpdateMode.....	798						

DioCl1MultiPwmOut_setDutyCycle

Syntax	<pre>UInt32 DioCl1MultiPwmOut_setDutyCycle(DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut, Float64* PwmDutyCycle)</pre>
Include file	IoDrvDioClass1MultiPwmOut.h
Purpose	To set the duty cycles of the multichannel PWM signals to be output at DIO Class 1 channels.
Description	<p>The duty cycles are defined as the ratio between T_{High} and T_{Period}. For each output channel of the channel group you can specify an initial value for the duty cycle or you can update the duty cycle during run time of the real-time application.</p> <p>The resolution of the duty cycles depends on the specified period value.</p> <p>The resolution for time intervals within the PWM signal generator unit is 10 ns.</p> <p>The setting does not take effect until you call DioCl1MultiPwmOut_apply during the initialization phase or DioCl1MultiPwmOut_write during run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.</p>

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the multichannel PWM output that you specified by using `DioCl1MultiPwmOut_create`, you must use `DioCl1MultiPwmOut_start`.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using `DioCl1MultiPwmOut_create`.

PwmDutyCycle Lets you specify an array with the duty cycles of the PWM signals to be generated in the range 0.0 ... 1.0, i.e., 0% ... 100%.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_setPeriod.....	790
DioCl1MultiPwmOut_start.....	799
DioCl1MultiPwmOut_vwrite.....	801

DioCl1MultiPwmOut_setEventDelay

Syntax

```
UInt32 DioCl1MultiPwmOut_setEventDelay(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    Float64 EventDelay)
```

Include file	<code>IoDrvDioClass1MultiPwmOut.h</code>						
Purpose	To set the delay for events related to DIO Class 1 PWM multichannel signal generation.						
Description	<p>The function is used to delay the generation of events that you specified by using <code>DioCl1MultiPwmOut_setInterruptMode</code> and <code>DioCl1MultiPwmOut_setTriggerMode</code>.</p> <p>The delay is the time interval between the occurrence of the event and the generation of the event signal (interrupt or trigger signal).</p> <p>The setting does not take effect until you call <code>DioCl1MultiPwmOut_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Multichannel PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using <code>DioCl1MultiPwmOut_create</code>.</p> <p>EventDelay Lets you specify the time interval by which the generation of an event is delayed in seconds in the range 0 ... 1.34 s. You can specify the value in steps of 10 ns.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						
Related topics	References						
	<table border="0"> <tr> <td>DioCl1MultiPwmOut_apply.....</td><td>772</td></tr> <tr> <td>DioCl1MultiPwmOut_create.....</td><td>774</td></tr> <tr> <td>DioCl1MultiPwmOut_setEventDownsample.....</td><td>783</td></tr> </table>	DioCl1MultiPwmOut_apply	772	DioCl1MultiPwmOut_create	774	DioCl1MultiPwmOut_setEventDownsample	783
DioCl1MultiPwmOut_apply	772						
DioCl1MultiPwmOut_create	774						
DioCl1MultiPwmOut_setEventDownsample	783						

DioCl1MultiPwmOut_setInterruptMode.....	784
DioCl1MultiPwmOut_setTriggerMode.....	796

DioCl1MultiPwmOut_setEventDownsample

Syntax

```
UInt32 DioCl1MultiPwmOut_setEventDownsample(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 DownsamplingValue)
```

Include file

`IoDrvDioClass1MultiPwmOut.h`

Purpose

To set the downsampling for events related to DIO Class 1 PWM multichannel signal generation.

Description

The function is used to reduce the number of generated events that you specified by using `DioCl1MultiPwmOut_setInterruptMode` and `DioCl1MultiPwmOut_setTriggerMode`.

The setting does not take effect until you call `DioCl1MultiPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using `DioCl1MultiPwmOut_create`.

DownsamplingValue Lets you specify the downsampling value for event generation (interrupt or trigger signal) in the range 1 ... 256.

The downsampling value specifies whether to generate an event on each occurrence of the event condition (EventDownsampling = 1, default) or only on each n-th occurrence, where n is the specified downsampling value.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_create.....	774
DioCl1MultiPwmOut_setEventDelay.....	781
DioCl1MultiPwmOut_setInterruptMode.....	784
DioCl1MultiPwmOut_setTriggerMode.....	796

DioCl1MultiPwmOut_setInterruptMode

Syntax

```
UInt32 DioCl1MultiPwmOut_setInterruptMode(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 InterruptMode)
```

Include file

`IoDrvDioClass1MultiPwmOut.h`

Purpose

To specify the interrupt mode for the generation of multichannel PWM output signals.

Description

This function is used to generate interrupts at certain points during the period of the output signals. You can specify to generate interrupts at the period's beginning or, if the alignment mode is set to center-alignment, at the period's middle or at both times. You can set the alignment mode via `DioCl1MultiPwmOut_setAlignmentMode`.

If you want to configure a combination of interrupts and trigger signals, you must configure the trigger mode in addition to the interrupt mode by using `DioCl1MultiPwmOut_setTriggerMode`.

The setting does not take effect until you call `DioCl1MultiPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using [DioCl1MultiPwmOut_create](#).

InterruptMode Lets you specify the interrupt mode for multichannel PWM signal generation. The following symbols can be combined by the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_NO_INTERRUPT	No interrupt generation (default).
DIO_CLASS1_INT_BEGIN_PERIOD	Specifies to generate interrupts at the beginning of a period.
DIO_CLASS1_INT_MID_PERIOD	For center-aligned PWM alignment mode only. Specifies to generate interrupts at the middle of a period.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_create.....	774
DioCl1MultiPwmOut_disableInterrupts.....	776
DioCl1MultiPwmOut_enableInterrupts.....	777
DioCl1MultiPwmOut_setIoIntVector.....	787

DioCl1MultiPwmOut_setInvertingMode

Syntax

```
UInt32 DioCl1MultiPwmOut_setInvertingMode(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 InvertingMode)
```

Include file	<code>IoDrvDioClass1MultiPwmOut.h</code>						
Purpose	To specify the inverting mode for the generation of multichannel PWM output signals.						
Description	<p>This function is used to enable the additional generation of inverted PWM signals. The number of required channels is then automatically doubled.</p> <p>For more information on the inverting mode, refer to Multichannel PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p> <p>The setting does not take effect until you call <code>DioCl1MultiPwmOut_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Multichannel PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using <code>DioCl1MultiPwmOut_create</code>.</p> <p>InvertingMode Lets you specify the inverting mode of the multichannel PWM signal generation.</p>						
<table border="1" data-bbox="179 1320 1389 1457"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>DIO_CLASS1_NOT_INVERTED</td><td>Specifies to generate non-inverted signals only (default).</td></tr> <tr> <td>DIO_CLASS1_INVERTED</td><td>Specifies to generate inverted signals additionally.</td></tr> </tbody> </table>	Symbol	Meaning	DIO_CLASS1_NOT_INVERTED	Specifies to generate non-inverted signals only (default).	DIO_CLASS1_INVERTED	Specifies to generate inverted signals additionally.	
Symbol	Meaning						
DIO_CLASS1_NOT_INVERTED	Specifies to generate non-inverted signals only (default).						
DIO_CLASS1_INVERTED	Specifies to generate inverted signals additionally.						
Return value	The function returns an error code.						
<table border="1" data-bbox="560 1579 1389 1717"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR != IOLIB_NO_ERROR</td><td>The function was successfully completed. The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR != IOLIB_NO_ERROR	The function was successfully completed. The function was not completed.			
Error Code	Meaning						
IOLIB_NO_ERROR != IOLIB_NO_ERROR	The function was successfully completed. The function was not completed.						

Related topics**References**

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_create.....	774

DioCl1MultiPwmOut_setIoIntVector

Syntax

```
UInt32 DioCl1MultiPwmOut_SetIoIntVector(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 InterruptSelect,
    IolibTToIntHandler InterruptHandler)
```

Include file

IoDrvDioClass1MultiPwmOut.h

Purpose

To set the interrupt handler for processing interrupts from multichannel PWM signal generation on DIO Class 1 output channels.

Description

This function is used to register the function that handles the generated interrupts specified by using **DioCl1MultiPwmOut_SetInterruptMode**. For each specified condition for interrupt generation (beginning or middle of the period), you can define separate functions. If you want to handle all conditions with the same function, you can use the bitwise OR operator to combine the predefined symbols of the **InterruptSelect** parameter.

Note

The condition specified in the **InterruptSelect** parameter must be a part of the conditions specified for the **DioCl1MultiPwmOut_SetInterruptMode** function.

The setting does not take effect until you call **DioCl1MultiPwmOut_apply** during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using **DioCl1MultiPwmOut_create**.

InterruptSelect Lets you select the condition the given interrupt handler is to be associated with. The symbols can be combined by using the bitwise OR operator.

Symbol	Meaning
DIO_CLASS1_INT_BEGIN_PERIOD	Specifies that the handler is to process interrupts generated at the beginning of the period.
DIO_CLASS1_INT_MID_PERIOD	Specifies that the handler is to process interrupts generated at the middle of the period.

InterruptHandler Lets you specify the address of the function which is to handle the generated interrupt.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_create.....	774
DioCl1MultiPwmOut_disableInterrupts.....	776
DioCl1MultiPwmOut_enableInterrupts.....	777
DioCl1MultiPwmOut_setInterruptMode.....	784

DioCl1MultiPwmOut_setOutputHighZ

Syntax

```
UInt32 DioCl1MultiPwmOut_setOutputHighZ(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 HighZStateOnOff)
```

Include file

IoDrvDioClass1MultiPwmOut.h

Purpose To set the high impedance state of the DIO Class 1 multichannel PWM output channels.

Description You can force the output channel of the driver object to the high impedance state by setting the high impedance state to *On*. This is useful for termination purposes. If you want the real-time application to control the output levels, you can release the high impedance state again.

The function is intended to be called during the run time of the real-time application.

The setting does not take effect until you call `DioCl1MultiPwmOut_start` or `DioCl1MultiPwmOut_write` during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the multichannel PWM output that you specified by using `DioCl1MultiPwmOut_create`, you must use `DioCl1MultiPwmOut_start`.

I/O mapping For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

<p>pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using <code>DioCl1MultiPwmOut_create</code>.</p> <p>HighZStateOnOff Lets you specify if outputs are to be set to the high impedance state or if outputs are to be released from the high impedance state.</p>

Symbol	Meaning
DIO_CLASS1_HIGH_Z_OFF	Releases the high impedance state.
DIO_CLASS1_HIGH_Z_ON	Sets outputs to the high impedance state.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_setSignalVoltage.....	792
DioCl1MultiPwmOut_start.....	799
DioCl1MultiPwmOut_write.....	801

DioCl1MultiPwmOut_setPeriod

Syntax

```
UInt32 DioCl1MultiPwmOut_setPeriod(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    Float64 PwmPeriod)
```

Include file

IoDrvDioClass1MultiPwmOut.h

Purpose

To set the period of the multichannel PWM signals to be output at DIO Class 1 channels.

Description

The setting does not take effect until you call **DioCl1MultiPwmOut_apply** during the initialization phase or **DioCl1MultiPwmOut_write** during run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller. During the initialization phase an error message is output additionally.

Note

All digital outputs of the DIO Class 1 unit are in a high impedance state after reset. To enable the multichannel PWM output that you specified by using **DioCl1MultiPwmOut_create**, you must use **DioCl1MultiPwmOut_start**.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts](#) ([MicroLabBox Hardware Installation and Configuration](#)).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\)](#) ([MicroLabBox Features](#)).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using [DioCl1MultiPwmOut_create](#).

PwmPeriod Lets you specify the period of the PWM signals to be generated in seconds in the range 100 ns ... 1.34 s. You can specify the value in steps of 10 ns.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_setDutyCycle.....	780
DioCl1MultiPwmOut_start.....	799
DioCl1MultiPwmOut_vwrite.....	801

DioCl1MultiPwmOut_setRisingEdgeDelay

Syntax

```
UInt32 DioCl1MultiPwmOut_setRisingEdgeDelay(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    Float64 RisingEdgeDelay)
```

Include file

`IoDrvDioClass1MultiPwmOut.h`

Purpose

To specify a delay for rising edges of the generated multichannel PWM output signals.

Description	<p>The function affects all output channels that are controlled by the specified driver object.</p> <p>The setting does not take effect until you call <code>DioCl1MultiPwmOut_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Multichannel PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p>pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using <code>DioCl1MultiPwmOut_create</code>.</p> <p>RisingEdgeDelay Lets you specify the delay for rising edges in seconds in the range 0 ... 655 µs with a resolution of 10 ns. The default is 0 s.</p>						
Return value	<p>The function returns an error code.</p> <table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td><td>The function was successfully completed.</td></tr> <tr> <td>!= IOLIB_NO_ERROR</td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics**References**

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_create.....	774

DioCl1MultiPwmOut_setSignalVoltage

Syntax

```
UInt32 DioCl1MultiPwmOut_setSignalVoltage(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 SignalVoltage)
```

Include file	<code>IoDrvDioClass1MultiPwmOut.h</code>								
Purpose	To specify the voltage level used for the DIO Class 1 multichannel PWM output channels.								
Description	<p>The output voltage level will be applied to the PWM output channels that are controlled by the specified DioCl1BcPwmOut driver object.</p> <p>The setting does not take effect until you call <code>DioCl1MultiPwmOut_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>								
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Multichannel PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>								
Parameters	<p>pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using <code>DioCl1MultiPwmOut_create</code>.</p> <p>SignalVoltage Lets you select the voltage level used for the output signals.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>DIO_CLASS1_SIGNAL_2_5_V</code></td><td>Specifies 2.5 V as the output voltage level (default).</td></tr> <tr> <td><code>DIO_CLASS1_SIGNAL_3_3_V</code></td><td>Specifies 3.3 V as the output voltage level.</td></tr> <tr> <td><code>DIO_CLASS1_SIGNAL_5_0_V</code></td><td>Specifies 5.0 V as the output voltage level.</td></tr> </tbody> </table>	Symbol	Meaning	<code>DIO_CLASS1_SIGNAL_2_5_V</code>	Specifies 2.5 V as the output voltage level (default).	<code>DIO_CLASS1_SIGNAL_3_3_V</code>	Specifies 3.3 V as the output voltage level.	<code>DIO_CLASS1_SIGNAL_5_0_V</code>	Specifies 5.0 V as the output voltage level.
Symbol	Meaning								
<code>DIO_CLASS1_SIGNAL_2_5_V</code>	Specifies 2.5 V as the output voltage level (default).								
<code>DIO_CLASS1_SIGNAL_3_3_V</code>	Specifies 3.3 V as the output voltage level.								
<code>DIO_CLASS1_SIGNAL_5_0_V</code>	Specifies 5.0 V as the output voltage level.								
Return value	The function returns an error code.								
	<table border="1"> <thead> <tr> <th>Error Code</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td><td>The function was successfully completed.</td></tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td><td>The function was not completed.</td></tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.		
Error Code	Meaning								
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.								
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.								
Related topics	References								
	<table border="1"> <tbody> <tr> <td><code>DioCl1MultiPwmOut_apply.....</code></td><td>772</td></tr> <tr> <td><code>DioCl1MultiPwmOut_create.....</code></td><td>774</td></tr> </tbody> </table>	<code>DioCl1MultiPwmOut_apply.....</code>	772	<code>DioCl1MultiPwmOut_create.....</code>	774				
<code>DioCl1MultiPwmOut_apply.....</code>	772								
<code>DioCl1MultiPwmOut_create.....</code>	774								

DioCl1MultiPwmOut_setOutputHighZ.....	788
---------------------------------------	-----

DioCl1MultiPwmOut_setTriggerLineOut

Syntax

```
UInt32 DioCl1MultiPwmOut_setTriggerLineOut(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 TriggerLineOut)
```

Include file

`IoDrvDioClass1MultiPwmOut.h`

Purpose

To select the trigger line used for trigger signal generation.

Description

A trigger line is required to transmit the generated trigger signal to the consumer on the I/O board. You must specify the trigger line that the consumer listens to by using the relevant `xxx_setTriggerLineIn` function. A trigger line can be allocated only once.

Before you can use trigger signals, you must set the trigger mode via `DioCl1MultiPwmOut_setTriggerMode`. With `DioCl1MultiPwmOut_setTriggerLineOutStatus` you enable or disable the trigger line.

The setting does not take effect until you call `DioCl1MultiPwmOut_apply` during the initialization phase.

If an error is detected, the first instance of the error is returned to the caller and an error message is output.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using `DioCl1MultiPwmOut_create`.

TriggerLineOut Lets you specify the number of the trigger line to be used for transmitting the trigger signal in the range 1 ... 16.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_1	Specifies trigger line 1.
...	...
DIO_CLASS1_TRIGGER_LINE_16	Specifies trigger line 16.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_apply.....	727
DioCl1BcPwmOut_create.....	728
DioCl1MultiPwmOut_setTriggerLineOutStatus.....	795
DioCl1MultiPwmOut_setTriggerMode.....	796

DioCl1MultiPwmOut_setTriggerLineOutStatus

Syntax

```
UInt32 DioCl1MultiPwmOut_setTriggerLineOutStatus(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 TriggerLineStatus)
```

Include file

IoDrvDioClass1MultiPwmOut.h

Purpose

To set the status of the trigger line.

Description

With this function, you can enable or disable the trigger line that you specified before by using **DioCl1MultiPwmOut_setTriggerLineOut**.

The function is intended to be called during the initialization phase of the real-time application or during the run time of the application.

The setting does not take effect until you call **DioCl1MultiPwmOut_start** during the initialization phase or during a run-to-stop or a stop-to-run transition of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using **DioCl1MultiPwmOut_create**.

TriggerLineStatus Lets you specify the status of the trigger line.

Symbol	Meaning
DIO_CLASS1_TRIGGER_LINE_DISABLE	Disables the trigger line.
DIO_CLASS1_TRIGGER_LINE_ENABLE	Enables the trigger line.

Return value

The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics**References**

DioCl1BcPwmOut_create.....	728
DioCl1MultiPwmOut_setTriggerLineOut.....	794
DioCl1MultiPwmOut_write.....	801

[DioCl1MultiPwmOut_setTriggerMode](#)**Syntax**

```
UInt32 DioCl1MultiPwmOut_setTriggerMode(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut,
    UInt32 TriggerMode)
```

Include file	<code>IoDrvDioClass1MultiPwmOut.h</code>
Purpose	To specify the trigger mode for the generation of multichannel PWM output signals.
Description	<p>This function is used to generate trigger signals related to the period of the output signals. You can select to generate trigger signals at the period's beginning, its middle or at both times.</p> <p>Additionally, you must specify the trigger line for the generated trigger signal via <code>DioCl1MultiPwmOut_setTriggerLineOut</code>. The I/O unit that is to consume the trigger signal has to specify this trigger line by using its related <code>xxx_setTriggerLineIn</code> function.</p> <p>If you want to configure a combination of trigger signals and interrupts, you must configure the interrupt mode in addition to the trigger mode by using <code>DioCl1MultiPwmOut_setInterruptMode</code>.</p> <p>The setting does not take effect until you call <code>DioCl1MultiPwmOut_apply</code> during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Multichannel PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>
Parameters	<p>pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using <code>DioCl1MultiPwmOut_create</code>.</p> <p>TriggerMode Lets you specify the trigger mode for the PWM signal generation. The following symbols can be combined by the bitwise OR operator.</p>
Symbol	Meaning
DIO_CLASS1_NO_TRIGGER	No trigger generation (default).
DIO_CLASS1_TRIG_BEGIN_PERIOD	Specifies to generate trigger events at the beginning of a period.
DIO_CLASS1_TRIG_MID_PERIOD	Specifies to generate trigger events at the middle of a period.

Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>IOLIB_NO_ERROR</td> <td>The function was successfully completed.</td> </tr> <tr> <td>!= IOLIB_NO_ERROR</td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	IOLIB_NO_ERROR	The function was successfully completed.	!= IOLIB_NO_ERROR	The function was not completed.
Error Code	Meaning						
IOLIB_NO_ERROR	The function was successfully completed.						
!= IOLIB_NO_ERROR	The function was not completed.						

Related topics	References						
	<table> <tr> <td>DioCl1MultiPwmOut_apply.....</td> <td>772</td> </tr> <tr> <td>DioCl1MultiPwmOut_create.....</td> <td>774</td> </tr> <tr> <td>DioCl1MultiPwmOut_setTriggerLineOut.....</td> <td>794</td> </tr> </table>	DioCl1MultiPwmOut_apply.....	772	DioCl1MultiPwmOut_create.....	774	DioCl1MultiPwmOut_setTriggerLineOut.....	794
DioCl1MultiPwmOut_apply.....	772						
DioCl1MultiPwmOut_create.....	774						
DioCl1MultiPwmOut_setTriggerLineOut.....	794						

DioCl1MultiPwmOut_setUpdateMode

Syntax	<pre>UInt32 DioCl1MultiPwmOut_setUpdateMode(DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut, UInt32 UpdateMode)</pre>
Include file	<code>IoDrvDioClass1MultiPwmOut.h</code>
Purpose	To set the update mode of the DIO Class 1 multichannel PWM output channels.
Description	<p>The function is used to select the update behavior that is applied when you change the PWM period or duty cycle. The available update modes depend on the alignment mode that you can specify via DioCl1MultiPwmOut_setAlignmentMode.</p> <p>The setting does not take effect until you call DioCl1MultiPwmOut_apply during the initialization phase.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Multichannel PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using [DioCl1MultiPwmOut_create](#).

UpdateMode Lets you set the update mode for the signal generation. The possible settings depend on the alignment mode of the generated signals. The setting is relevant for all specified channels. For details on the update mode, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Symbol	Meaning
DIO_PWM_ASYNC_UPDATE	For edge-aligned PWM alignment mode only. Specifies to update changes asynchronously.
DIO_PWM_SYNC_UPDATE	For edge-aligned PWM alignment mode only. Specifies to update changes synchronously (default).
DIO_PWM_BEGIN_UPDATE	For center-aligned PWM alignment mode only. Specifies to update changes at the beginning of the next period.
DIO_PWM_MID_UPDATE	For center-aligned PWM alignment mode only. Specifies to update changes at the middle of the next period.
DIO_PWM_ANY_UPDATE	For center-aligned PWM alignment mode only. Specifies to update changes at the beginning and at the middle of the next period.

Return value The function returns an error code.

Error Code	Meaning
IOLIB_NO_ERROR	The function was successfully completed.
!= IOLIB_NO_ERROR	The function was not completed.

Related topics

References

DioCl1MultiPwmOut_apply.....	772
DioCl1MultiPwmOut_create.....	774
DioCl1MultiPwmOut_setAlignmentMode.....	779

DioCl1MultiPwmOut_start

Syntax

```
UInt32 DioCl1MultiPwmOut_start(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut)
```

Include file	<code>IoDrvDioClass1MultiPwmOut.h</code>						
Purpose	To start generating multichannel PWM signals via DIO Class 1 output channels.						
Description	<p>This function is used to start the specified DioCl1MultiPwmOut driver. The driver activates the following points:</p> <ul style="list-style-type: none"> ▪ It activates the multichannel PWM signal generation. ▪ It activates the digital output channels. <p>The digital output channels are enabled to generate the multichannel PWM output signals. The output channels stay disabled if you call <code>DioCl1MultiPwmOut_setOutputHighZ(DIO_CLASS1_HIGH_Z_ON)</code> before.</p> <p>The function is intended to be called during the run time of the real-time application.</p> <p>If an error is detected, the first instance of the error is returned to the caller and an error message is output.</p>						
I/O mapping	<p>For details on MicroLabBox's I/O connectors, refer to Connector Pinouts (MicroLabBox Hardware Installation and Configuration).</p> <p>For more information on the I/O mapping, refer to Multichannel PWM Signal Generation (DIO Class 1) (MicroLabBox Features).</p>						
Parameters	<p><code>pDioCl1MultiPwmOut</code> Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using <code>DioCl1MultiPwmOut_create</code>.</p>						
Return value	The function returns an error code.						
	<table border="1"> <thead> <tr> <th>Error Code</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td><code>IOLIB_NO_ERROR</code></td> <td>The function was successfully completed.</td> </tr> <tr> <td><code>!= IOLIB_NO_ERROR</code></td> <td>The function was not completed.</td> </tr> </tbody> </table>	Error Code	Meaning	<code>IOLIB_NO_ERROR</code>	The function was successfully completed.	<code>!= IOLIB_NO_ERROR</code>	The function was not completed.
Error Code	Meaning						
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.						
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.						
Related topics	References						
	<table border="0"> <tr> <td><code>DioCl1MultiPwmOut_apply.....</code></td> <td>772</td> </tr> <tr> <td><code>DioCl1MultiPwmOut_create.....</code></td> <td>774</td> </tr> <tr> <td><code>DioCl1MultiPwmOut_vwrite.....</code></td> <td>801</td> </tr> </table>	<code>DioCl1MultiPwmOut_apply.....</code>	772	<code>DioCl1MultiPwmOut_create.....</code>	774	<code>DioCl1MultiPwmOut_vwrite.....</code>	801
<code>DioCl1MultiPwmOut_apply.....</code>	772						
<code>DioCl1MultiPwmOut_create.....</code>	774						
<code>DioCl1MultiPwmOut_vwrite.....</code>	801						

DioCl1MultiPwmOut_write

Syntax

```
UInt32 DioCl1MultiPwmOut_write(
    DioCl1MultiPwmOutSDrvObject *pDioCl1MultiPwmOut)
```

Include file

`IoDrvDioClass1MultiPwmOut.h`

Purpose

To update the settings of the multichannel PWM signal generation for DIO Class 1 output channels during run time.

Description

If you have used one of the following functions during run time, their settings will not be updated on the hardware until you call `DioCl1MultiPwmOut_write`:

- `DioCl1MultiPwmOut_setPeriod`
- `DioCl1MultiPwmOut_setDutyCycle`
- `DioCl1MultiPwmOut_setOutputHighZ`

The specified digital output channels must be enabled before via `DioCl1MultiPwmOut_start`.

The function is intended to be called during the run time of the real-time application.

If an error is detected, the first instance of the error is returned to the caller.

I/O mapping

For details on MicroLabBox's I/O connectors, refer to [Connector Pinouts \(MicroLabBox Hardware Installation and Configuration\)](#).

For more information on the I/O mapping, refer to [Multichannel PWM Signal Generation \(DIO Class 1\) \(MicroLabBox Features\)](#).

Parameters

pDioCl1MultiPwmOut Lets you specify the target PWM multichannel signal generation via the corresponding DioCl1MultiPwmOut driver object. The DioCl1MultiPwmOut driver object had to be created before by using `DioCl1MultiPwmOut_create`.

Return value

The function returns an error code.

Error Code	Meaning
<code>IOLIB_NO_ERROR</code>	The function was successfully completed.
<code>!= IOLIB_NO_ERROR</code>	The function was not completed.

Related topics

References

DioCl1MultiPwmOut_create.....	774
DioCl1MultiPwmOut_disableInterrupts.....	776
DioCl1MultiPwmOut_enableInterrupts.....	777
DioCl1MultiPwmOut_setDutyCycle.....	780
DioCl1MultiPwmOut_setOutputHighZ.....	788
DioCl1MultiPwmOut_setPeriod.....	790
DioCl1MultiPwmOut_start.....	799

Slave CAN Access Functions

Where to go from here

Information in this section

Basics on Slave CAN Access Functions	804
Basics on the communication between the master processor and the slave CAN subsystem.	
Data Structures for CAN	807
Information on internal data structures.	
Initialization	816
Initializing the CAN controller.	
CAN Channel Handling	818
Information on setting up the CAN channels.	
CAN Message Handling	829
Information on handling all types of CAN messages.	
CAN Service Functions	870
Information on updating the CAN service structure.	
CAN Subinterrupt Handling	875
Information on defining subinterrupts caused by certain events.	
Utilities	877
Information on setting the time base, clearing CAN data on the master, and reading the error code.	
Examples of Using CAN	880
Examples of how to use the CAN functions.	

Basics on Slave CAN Access Functions

Introduction	Provides basics on the communication principles between the master processor and the slave CAN subsystem, and on the CAN error message types.
---------------------	---

Where to go from here	Information in this section
	Basic Principles of Master-Slave Communication..... 804 The slave access functions are used to control the slave CAN subsystem by the master and exchange data between master and slave.
	CAN Error Message Types..... 805 The functions of the CAN environment report error, warning, and information messages if a problem occurs.

Basic Principles of Master-Slave Communication

Introduction	The master processor uses slave access functions to control the slave CAN subsystem and exchange data with it.
---------------------	--

Note

You have to initialize the communication between the master and the slaves. Refer to [can_tp1_communication_init](#) on page 816.

Communication process	<ul style="list-style-type: none">▪ The master application initializes the required slave functions based on the CAN controller.▪ The message register functions write all required values to the appropriate handle, e.g. (CAN_TP1_canMsg). The appropriate request and read functions get the information from this handle later on.▪ To perform a read operation, the master processor requests that the previously registered slave function be carried out. The slave then performs the required functions independently and writes the results back to the dual-port memory. If more than one function is required simultaneously – for example, as a result of different tasks on the processor – priorities must be considered.▪ The master processor application reads/writes the input/output data from/to the slave.
------------------------------	--

Note

The master processor reads the slave results from the dual-port memory in the order in which they occur, and then reads them into a buffer, regardless of whether a particular result is needed. The read functions copy data results from the buffer into the processor application variables.

Function classes

Slave applications are based on communication functions that are divided into separate classes as follows:

- *Initialization functions* initialize the slave functions.
- *Register functions* make the slave functions known to the slave.
- *Request functions* require that the previously registered slave function be carried out by the slave.
- *Read functions* fetch data from the dual-port memory and convert or scale the data, if necessary.
- *Write functions* convert or scale the data if necessary and write them into the dual-port memory.

Error handling

When an error occurs with initialization or register functions, an error message appears from the global message module. Then the program ends.

Request, read, and write functions return an error code. The application can then handle the error code.

Communication channels and priorities

This communication method, along with the command table and the transfer buffer, can be initialized in parallel for the statically defined communication channels with fixed priorities (0 ... 6). Like communication buffers, each communication channel has access to memory space in the dual-port memory so that slave error codes can be transferred.

Related topics**Basics**

[Basics on the RTI CAN Blockset \(RTI CAN Blockset Reference\)](#) 
[CAN Support \(MicroLabBox Features\)](#) 

CAN Error Message Types

Introduction

The functions of the CAN environment report error, warning, and information messages if a problem occurs. These messages are displayed by the **Message**

Viewer of the experiment software. The message consists of an error number, the function name, the module number and the message text. For example:

Error[121]: can_tp1_channel_init (6,...) baudrate: too low (min. 10 kBaud)!

Message Number	Message Type
100 ... 249	Error
250 ... 349	Warning
400 ... 500	Information

Related topics**References**

[can_tp1_channel_init](#).....818

Data Structures for CAN

Introduction

The data structures provide information on channels, services, and messages to be used by other functions. Using CAN RTLib functions, you access the structures *automatically*. You do not have to access them explicitly in your application.

Where to go from here

Information in this section

can_tp1_canChannel	807
The <code>can_tp1_canChannel</code> structure contains information on the CAN channel capabilities.	
can_tp1_canService	809
The structure contains information on the CAN services. It provides information on errors and status information.	
can_tp1_canMsg	812
The <code>can_tp1_canMsg</code> structure contains information on the CAN message capabilities.	

Information in other sections

CAN Error Message Types	805
The functions of the CAN environment report error, warning, and information messages if a problem occurs.	

[can_tp1_canChannel](#)

Purpose

The `can_tp1_canChannel` structure contains information on the CAN channel capabilities.

Syntax

```
typedef struct
{
    UInt32 module_addr;
    UInt32 module;
    Int32 index;
    UInt32 channel;
    UInt32 btr0;
    UInt32 btr1;
    UInt32 frequency;
    UInt32 mb15_format;
    UInt32 busoff_int_number;
} can_tp1_canChannel;
```

Include file

CanTp1.h

Members**module_addr** Module address of the CAN module**module** The CAN module address is provided by the function `can_tp1_channel_init`. This parameter is read-only.**index** Table index allocated by the message register function.**channel** Number of the used CAN channel. This parameter is provided by the function `can_tp1_channel_init`. This parameter is read-only.**btr0** Value of Bit Timing Register 0. This parameter is provided by the function `can_tp1_channel_init`. This parameter is read-only.**btr1** Value of Bit Timing Register 1. This parameter is provided by the function `can_tp1_channel_init`. This parameter is read-only.**frequency** Frequency of the CAN controller. This parameter is provided by the function `can_tp1_channel_init`. This parameter is read-only.**mb15_format** Format of mailbox 15. Mailbox 15 is a double-buffered receive unit of the CAN. Use this mailbox for the message type most frequently used in your application. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_STD	11-bit standard format, CAN 2.0A
CAN_TP1_EXT	29-bit extended format, CAN 2.0B

This parameter is provided by the function `can_tp1_channel_init`. This parameter is read-only.

busoff_int_number Subinterrupt generated when the CAN channel goes bus off. This parameter is provided by the function `can_tp1_channel_init`. This parameter is read-only.

can_tp1_canService

Purpose	The <code>can_tp1_canService</code> structure contains information on the CAN service. The CAN service provides information on errors and status information (see the <code>type</code> parameter).
----------------	---

Syntax

```
typedef struct
{
    UInt32 busstatus;
    UInt32 stdmask;
    UInt32 extmask;
    UInt32 msg_mask15;
    UInt32 tx_ok;
    UInt32 rx_ok;
    UInt32 crc_err;
    UInt32 ack_err;
    UInt32 form_err;
    UInt32 stuffbit_err;
    UInt32 bit1_err;
    UInt32 bit0_err;
    UInt32 rx_lost;
    UInt32 data_lost;
    UInt32 mailbox_err;
    UInt32 data0;
    UInt32 data1;
    UInt16 txqueue_overflowcnt_std;
    UInt16 txqueue_overflowcnt_ext;
    UInt32 module;
    UInt32 queue;
    UInt32 type;
    Int32 index;
} can_tp1_canService;
```

Include file	CanTp1.h
---------------------	----------

Members	data0 Contains returned data from the function <code>can_tp1_service_read</code> . data1 Contains returned data from the function <code>can_tp1_service_read</code> .
----------------	--

Note

For each service, the structure provides its own member. For the meaning of the services, refer to the `type` parameter. The members `data0` and `data1` remain in the structure for compatibility reasons.

module The CAN module is provided by the function `can_tp1_service_register`. This parameter is read-only.

queue This parameter is provided by the function `can_tp1_service_register`. This parameter is read-only.

type Type of the service already allocated by the previously performed register function. Once a service is registered on the slave, it can deliver a value. The return value will be stored in the structure members `data0` and `data1`. This parameter is provided by the `can_tp1_service_register` function. This parameter is read-only.

Note

Start the CAN channel with the enabled status interrupt to use the following predefined services (see [can_tp1_channel_start](#) on page 821).

Predefined Symbol	Meaning
<code>CAN_TP1_SERVICE_TX_OK</code>	Number of successfully sent TX/RM/RQTX messages
<code>CAN_TP1_SERVICE_RX_OK</code>	Number of successfully received RX/RQRX messages
<code>CAN_TP1_SERVICE_CRC_ERR</code>	Number of CRC errors
<code>CAN_TP1_SERVICE_ACK_ERR</code>	Number of acknowledge errors
<code>CAN_TP1_SERVICE_FORM_ERR</code>	Number of format errors
<code>CAN_TP1_SERVICE_BIT1_ERR</code>	Number of Bit1 errors
<code>CAN_TP1_SERVICE_BIT0_ERR</code>	Number of Bit0 errors
<code>CAN_TP1_SERVICE_STUFFBIT_ERR</code>	Number of stuff bit errors

Note

It is not necessary to start the CAN channel with the enabled status interrupt if you are using only the following predefined services (see [can_tp1_channel_start](#) on page 821).

Predefined Symbol	Meaning
<code>CAN_TP1_SERVICE_RX_LOST</code>	Number of lost RX messages. The RX lost counter is incremented when a received message is overwritten in the receive mailbox before the message has been read.
<code>CAN_TP1_SERVICE_DATA_LOST</code>	Number of data lost errors. The data lost counter is incremented when the data of a message is overwritten before the data has been written to the communication queue.
<code>CAN_TP1_SERVICE_MAILBOX_ERR</code>	Number of mailbox errors. If a message to be sent cannot be assigned to a mailbox, the mailbox error counter is increased by one. For possible error reasons, see below.
<code>CAN_TP1_SERVICE_BUSSTATUS</code>	Status of the CAN controller. For the predefined values, see below.
<code>CAN_TP1_SERVICE_STDMASK</code>	Status of the global standard mask register
<code>CAN_TP1_SERVICE_EXTMASK</code>	Status of the global extended mask register
<code>CAN_TP1_SERVICE_MSG_MASK15</code>	Status of the message 15 mask register

Predefined Symbol	Meaning
CAN_TP1_SERVICE_TXQUEUE_OVERFLOW_COUNT	<p>Overflow counter of the transmit queue. The overflow counter (STD or XTD message format) is incremented when the queue is filled (64 messages) and a new message arrives. Depending on the <code>overrun_policy</code> parameter set with <code>can_tp1_msg_txqueue_init</code>, the new message overwrites the oldest message entry or is ignored.</p> <p>The overflow counters are 16-bit counters. The wraparound occurs after 65535 overflows.</p>

index Table index already allocated by the register function `can_tp1_service_register`. This parameter is read-only.

Parameter type

Additional information on the service functions provided by the type parameter:

CAN_TP1_SERVICE_MAILBOX_ERR Provides the number of mailbox errors. The following table describes possible error reasons and how to you can avoid these errors:

Error reason	Description	Workaround
All mailboxes are filled.	The messages are not removed from a mailbox fast enough.	Decrease the timeout value of all messages of the corresponding CAN channel and restart the application.
Conflict between two message IDs.	This error can occur if standard and extended messages are used on a CAN channel simultaneously. Check whether all messages are sent according to your requirements. It is not possible to remove remote messages temporarily from a mailbox. Check for a possible problem with a registered remote message.	<p>Try the first element of the following list. If the error counter still increases, try the next one:</p> <ul style="list-style-type: none"> ▪ Decrease the timeout value for messages with the same format as mailbox 14 – i.e., with the opposite format of mailbox 15 (refer to <code>can_tp1_channel_init</code>). ▪ Initialize the <code>mb15_format</code> parameter with the other format when calling <code>can_tp1_channel_init</code>. ▪ Choose different message IDs for messages of mailbox 14 format. ▪ Do not use standard and extended messages on one CAN channel simultaneously.

CAN_TP1_SERVICE_BUSSTATUS Provides bus status information; the following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_BUSOFF_STATE	The CAN channel disconnects itself from the CAN bus. Use <code>can_tp1_channel1_B0ff_return</code> to recover from the bus off state.
CAN_TP1_WARN_STATE	The CAN controller is still active. The CAN controller recovers from this state automatically.
CAN_TP1_ACTIVE_STATE	The CAN controller is active.

Note

After calling `can_tp1_channel_BOff_return`, the service `CAN_TP1_SERVICE_BUSSTATUS` will not return `CAN_TP1_BUSOFF_STATE`.

Example The following example shows you how to use the CAN service with an overflow counter:

```
can_tp1_canService* service;
UInt16 overflow;
...
service = can_tp1_service_register(
    txCh, CAN_TP1_SERVICE_TXQUEUE_OVERFLOW_COUNT);
...
can_tp1_service_request( service );
can_tp1_service_read( service );
overflow = service->txqueue_overflowcnt_std;
```

Related topics**References**

<code>can_tp1_channel_BOff_go</code>	824
<code>can_tp1_channel_BOff_return</code>	825
<code>can_tp1_channel_init</code>	818
<code>can_tp1_channel_start</code>	821
<code>can_tp1_msg_txqueue_init</code>	855
<code>can_tp1_service_read</code>	873
<code>can_tp1_service_register</code>	870

can_tp1_canMsg

Purpose

The `can_tp1_canMsg` structure contains information on the CAN message capabilities.

Syntax

```
typedef struct{
    Float64 timestamp;
    Float32 deltatime;
    Float32 delaytime;
    Int32 processed;
    UInt32 datalen;
    UInt32 data[8];
    UInt32 identifier;
    UInt32 format;
    UInt32 module;
    UInt32 queue;
    Int32 index;
    UInt32 msg_no;
    UInt32 type;
    UInt32 inform;
    UInt32 timecount;
    can_tp1_canChannel*canChannel;
    can_tp1_canService *msgService;
} can_tp1_canMsg;
```

Include file

CanTp1.h

Members**timestamp** This parameter contains the following values:

- For transmit or remote messages: The point in time the last message was successfully sent (given in seconds).
- For receive messages: The point in time the last message was received (given in seconds).

This parameter is updated by the function `can_tp1_msg_read` if the message was registered using the `inform` parameter `CAN_TP1_TIMECOUNT_INFO`.

deltatime Time difference in seconds between the old and the new timestamp

This parameter is updated by the function `can_tp1_msg_read` if the message was registered with the `inform` parameter `CAN_TP1_TIMECOUNT_INFO`.

Note

If several CAN identifiers are received with a single RX message, the `deltatime` parameter delivers useless values. For this reason, it is recommended to use the `deltatime` parameter only if one CAN identifier is received per registered CAN message.

delaytime Time difference between the update and the sending of a message (for TX, RQTX, and RM messages only). For cyclic sending, the delay time between the update and the sending of a message is used. For acyclic sending, the delay time between the trigger and the successful sending of a message is used. The valid range is 0.0 ... 100.0 seconds.

This parameter is updated by the function `can_tp1_msg_read` if the message was registered with the `inform` parameter `CAN_TP1_DELAYCOUNT_INFO`.

processed Processed flag of the message. This parameter is updated by the function `can_tp1_msg_read`. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_PROCESSED</code>	The message has been sent/received since the last execution call.
<code>CAN_TP1_NOT_PROCESSED</code>	The message has not been sent/received since the last execution call.

datalen Length of the data in the CAN message in bytes. This parameter is updated by the function `can_tp1_msg_read` if the message was registered with the `inform` parameter `CAN_TP1_DATA_INFO`.

data[8] Buffer for CAN message data. This data is updated by the function `can_tp1_msg_read` if the message was registered with the `inform` parameter `CAN_TP1_DATA_INFO`.

identifier Identifier of the message. This parameter is provided by the message register functions and is read-only.

format Specifies the message format. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_STD</code>	11-bit standard format, CAN 2.0A
<code>CAN_TP1_EXT</code>	29-bit extended format, CAN 2.0B

module Address of the registered message. This parameter is provided by the message register functions and is read-only.

queue Communication channel within the range of 0 ... 5. This parameter is provided by the message register functions and is read-only.

index Table index already allocated by the previously performed register function. This parameter is provided by the message register functions and is read-only.

msg_no Number of the message. This parameter is provided by the message register functions and is read-only.

type Type of the CAN message. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_TX</code>	Transmit message registered by <code>can_tp1_msg_tx_register</code>
<code>CAN_TP1_RX</code>	Receive message registered by <code>can_tp1_msg_rx_register</code>
<code>CAN_TP1_RM</code>	Remote message registered by <code>can_tp1_msg_rm_register</code>
<code>CAN_TP1_RQTX</code>	RQTX message registered by <code>can_tp1_msg_rqtx_register</code>
<code>CAN_TP1_RQRX</code>	RQRX message registered by <code>can_tp1_msg_rqrx_register</code>

This parameter is provided by the message register functions and is read-only.

inform Specifies the kind of information returned by the function `can_tp1_msg_read`. You have to register a message with the appropriate `inform` parameter to get the requested information. You can combine the predefined symbols with the logical operator OR. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_NO_INFO</code>	Returns no information.
<code>CAN_TP1_DATA_INFO</code>	Updates the data and datalen parameters (needed for receive and request (RQRX) messages).
<code>CAN_TP1_MSG_INFO</code>	Updates the message identifier and the message format for RM, RQ, TX, and RX messages.
<code>CAN_TP1_TIMECOUNT_INFO</code>	Updates the timestamp and the deltatime parameters.
<code>CAN_TP1_DELAYCOUNT_INFO</code>	Updates the delaytime parameter.

Note

If you modify the `inform` parameter after the message was registered, your message data will be corrupted.

This parameter is provided by the message register functions and is read-only.

timecount Internally used parameter. This parameter is read-only.

canChannel Pointer to the used `can_tp1_canChannel` structure where the message object is installed. This parameter is read-only. Refer to [can_tp1_canChannel](#) on page 807.

msgService Only used by the message processed functions to read the processed status (sent or received) of a message. This parameter is read-only.

Related topics

References

<code>can_tp1_msg_read.....</code>	862
<code>can_tp1_msg_rm_register.....</code>	844
<code>can_tp1_msg_rqrx_register.....</code>	841
<code>can_tp1_msg_rqtx_register.....</code>	837
<code>can_tp1_msg_rx_register.....</code>	834
<code>can_tp1_msg_tx_register.....</code>	830

Initialization

Introduction

Before you can use a CAN controller, you have to perform an initialization process that resets the slave DSP and sets up the communication channels between master and slave (parameter `queue`).

can_tp1_communication_init

Syntax

```
void can_tp_communication_init(
    const UInt32 ModuleAddr,
    const UInt32 bufferwarn)
```

Include file

`CanTp1.h`

Purpose

To initialize communication between the master and the slave on the `can_tp1` module.

Description

This function also initializes seven communication channels with fixed queues (0 ... 6) for the master-slave communication. The communication channel `QUEUE0` has the highest priority. The slave initializes the communication with the master itself and sends an acknowledgment code if the initialization was successful. If the master does not receive this acknowledgment code within one second, the program is aborted. The timer of the slave `can_tp1` is reset.

Parameters

ModuleAddr Specifies the address of the CAN module.

You can use the following predefined symbol:

Predefined Symbol	CAN Type1 Module Number (used by RTI)
<code>CAN_TP1_1_MODULE_ADDR</code>	1

bufferwarn Enables the bufferwarn subinterrupt. The subinterrupt handler is installed automatically. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_INT_DISABLE</code>	The bufferwarn subinterrupt is disabled.
<code>CAN_TP1_INT_ENABLE</code>	The bufferwarn subinterrupt is enabled.

Return value

None

Messages

The following messages are defined:

ID	Type	Message	Description
100	Error	can_tp1_communication_init(x,..) slave: no can_tp1 on moduleplace	The module can_tp1 was not found on the specified module address.
101	Error	can_tp1_communication_init(x,..) memory: allocation error on master	Memory allocation error. No free memory on the master.
104	Error	can_tp1_communication_init(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
105	Error	can_tp1_communication_init(x,..) subint: init failed by master	Master subinterrupt initialization failed. There is not enough memory available.
106	Error	can_tp1_communication_init(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second due to a wrong firmware version or a hardware failure.
107	Error	can_tp1_communication_init(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	can_tp1_communication_init(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data to a filled queue. To prevent this error, deactivate all messages with <code>can_tp1_msg_sleep</code> or <code>can_tp1_channel_all_sleep</code> when registering messages or services.
110	Error	can_tp1_communication_init(x,..) slave: No CAN_TP1 module available.	There is no CAN module at the given module address.
200	Error	can_tp1_communication_init(x,..) slave: not connected to HwiInterrupt	There might be a hardware failure or the initialization process is not correct.
400	Info	can_tp1_communication_init(x,..) CAN_TP1 booted from flash.	The slave on the can_tp1 module booted from flash memory.

Example

For examples, refer to:

- [Example of Handling Transmit and Receive Messages](#) on page 880
- [Example of Handling Request and Remote Messages](#) on page 881
- [Example of Using Subinterrupts](#) on page 883

Related topics**References**

<code>can_tp1_channel_all_sleep</code>	822
<code>can_tp1_channel_init</code>	818
<code>can_tp1_msg_sleep</code>	860

CAN Channel Handling

Introduction Provides information on handling CAN interfaces, called *CAN channels*.

Where to go from here	Information in this section
	can_tp1_channel_init818 To perform the basic initialization of the specified CAN channel, that is, to reset the CAN controller and set its baud rate.
	can_tp1_channel_start821 To complete the initialization and start the CAN channel referenced by the canCh pointer.
	can_tp1_channel_all_sleep822 To stop the transmission of all previously registered transmit, request transmission, and remote messages and the data transfer from all registered messages to the master processor.
	can_tp1_channel_all_wakeup823 To reactivate all messages that were deactivated by calling the functions <code>can_tp1_channel_all_sleep</code> and <code>can_tp1_msg_sleep</code> .
	can_tp1_channel_BOff_go824 To set the CAN channel to the bus off state. All bus operations performed by the CAN channel are canceled.
	can_tp1_channel_BOff_return825 To reset the slave can_tp1 CAN channel from the bus off state.
	can_tp1_channel_set826 To set a mask value or attribute for the specified CAN channel. Use this function to write the value to the specified CAN controller memory area.
	can_tp1_channel_txqueue_clear828 To clear the content of the transmit queues of the selected CAN channel.

can_tp1_channel_init

Syntax

```
can_tp1_canChannel* can_tp1_channel_init(
  const UInt32 ModuleAddr,
  const UInt32 channel,
  const UInt32 baudrate,
  const UInt32 mb15_format,
  const Int32 busoff_subinterrupt);
```

Include file	CanTp1.h
---------------------	----------

Purpose	To perform the basic initialization of the specified CAN channel, that is, to reset the CAN controller and set its baud rate.
----------------	---

Note

You have to call the `can_tp1_channel_start` function to complete the CAN channel initialization.

Description	If no error occurs, <code>can_tp1_channel_init</code> returns a pointer to the <code>can_tp1_canChannel</code> structure. If an interrupt is to be sent for the bus off state of the CAN controller, you have to specify a subinterrupt number and a subinterrupt handler.
--------------------	---

Parameters	<p>ModuleAddr Specifies the address of the CAN module. You can use the following predefined symbol:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th> <th>CAN Type1 Module Number (used by RTI)</th> </tr> </thead> <tbody> <tr> <td>CAN_TP1_1_MODULE_ADDR</td> <td>1</td> </tr> </tbody> </table> <p>channel Specifies the CAN channel within the range 0 ... 1.</p> <p>baudrate Specifies the baud rate of the CAN bus within the range 10 kBd ... 1 MBd.</p> <p>mb15_format Specifies the format for mailbox 15. Mailbox 15 is a double-buffered receive unit of the CAN. Use this mailbox for the message type most frequently used in your application. The following symbols are predefined:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CAN_TP1_STD</td> <td>11-bit standard format, CAN 2.0A</td> </tr> <tr> <td>CAN_TP1_EXT</td> <td>29-bit extended format, CAN 2.0B</td> </tr> </tbody> </table> <p>busoff_subinterrupt Specifies the Subinterrupt number for the bus off state. The valid range is 0 ... 14. Use the following predefined symbol to disable the bus off interrupt:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CAN_TP1_NO_SUBINT</td> <td>No interrupt for bus off</td> </tr> </tbody> </table>	Predefined Symbol	CAN Type1 Module Number (used by RTI)	CAN_TP1_1_MODULE_ADDR	1	Predefined Symbol	Meaning	CAN_TP1_STD	11-bit standard format, CAN 2.0A	CAN_TP1_EXT	29-bit extended format, CAN 2.0B	Predefined Symbol	Meaning	CAN_TP1_NO_SUBINT	No interrupt for bus off
Predefined Symbol	CAN Type1 Module Number (used by RTI)														
CAN_TP1_1_MODULE_ADDR	1														
Predefined Symbol	Meaning														
CAN_TP1_STD	11-bit standard format, CAN 2.0A														
CAN_TP1_EXT	29-bit extended format, CAN 2.0B														
Predefined Symbol	Meaning														
CAN_TP1_NO_SUBINT	No interrupt for bus off														
Return value	canChannel Pointer to the <code>can_tp1_canChannel</code>														

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	can_tp1_channel_init_advanced(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
104	Error	can_tp1_channel_init_advanced(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.
106	Error	can_tp1_channel_init_advanced(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second due to a wrong firmware version or a hardware failure.
107	Error	can_tp1_channel_init_advanced(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	can_tp1_channel_init_advanced(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data to a filled queue. To prevent this error, deactivate all messages with can_tp1_msg_sleep or can_tp1_channel_all_sleep when registering messages or services.
123	Error	can_tp1_channel_init_advanced(x,..) channel: use range 0..1!	Use a CAN channel within the range of 0 ... 1.
124	Error		The clock frequency of the CAN clock generator limited by is too low.
140	Error	can_tp1_channel_init_advanced(x,..) format: wrong format.	Only the symbols CAN_TP1_STD and CAN_TP1_EXT are allowed for the parameter mb15_format.
141	Error	can_tp1_channel_init_advanced(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range of 0 ... 14.

Example

For examples, refer to:

- [Example of Handling Transmit and Receive Messages](#) on page 880
- [Example of Handling Request and Remote Messages](#) on page 881
- [Example of Using Subinterrupts](#) on page 883

Related topics**References**

can_tp1_canChannel.....	807
can_tp1_channel_all_sleep.....	822
can_tp1_channel_all_wakeup.....	823
can_tp1_channel_BOff_go.....	824
can_tp1_channel_BOff_return.....	825
can_tp1_channel_set.....	826
can_tp1_channel_start.....	821
can_tp1_msg_rm_register.....	844
can_tp1_msg_rqrx_register.....	841

can_tp1_msg_rqtx_register.....	837
can_tp1_msg_rx_register.....	834
can_tp1_msg_sleep.....	860
can_tp1_msg_tx_register.....	830
can_tp1_service_register.....	870

can_tp1_channel_start

Syntax

```
void can_tp1_channel_start(
    const can_tp1_canChannel* canCh,
    const UInt32 status_int);
```

Include file

CanTp1.h

Purpose

To complete the initialization and start the CAN channel referenced by the canCh pointer.

Description

The CAN channel will change to the bus on state and the can_tp1 slave interrupts will be enabled. Use the returned handle from the function `can_tp1_channel_init` to call this function.

Parameters

canCh Specifies the pointer to the `can_tp1_canChannel` structure.

status_int Enables the status change interrupt; the following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_INT_DISABLE	No status interrupt will be generated.
CAN_TP1_INT_ENABLE	A status change interrupt can be generated when a CAN bus event is detected in the Status Register. A status change interrupt occurs on each successful reception or transmission on the CAN bus, regardless of whether the can_tp1 slave has configured a message object to receive that particular message identifier. This interrupt is useful to detect bus errors caused by physical layer issues, such as noise. In most applications, it is recommended to not set this bit. Because this interrupt occurs for each message, the can_tp1 would be unnecessarily burdened.

Return value

None

Messages

The following messages are defined:

ID	Type	Message	Description
104	Error	can_tp1_channel_start queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.

Example

For examples, refer to:

- [Example of Handling Transmit and Receive Messages](#) on page 880
- [Example of Handling Request and Remote Messages](#) on page 881
- [Example of Using Subinterrupts](#) on page 883

Related topics**References**

can_tp1_channel_init	818
--	-----

can_tp1_channel_all_sleep

Syntax

```
Int32 can_tp1_channel_all_sleep(
    const can_tp1_canChannel* canCh);
```

Include file

CanTp1.h

Purpose

To stop the transmission of all previously registered transmit, request transmission, and remote messages and the data transfer from all registered messages to the master processor.

Description

The messages are deactivated and set to sleep mode until they are reactivated by `can_tp1_channel_all_wakeup`.

Use the returned handle from the `can_tp1_channel_init` function to call this function.

Parameters

canCh Specifies the pointer to the `can_tp1_canChannel` structure.

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns CAN_TP1_NO_ERROR.

Example `can_tp1_channel_all_sleep(canCh);`

Related topics **References**

can_tp1_canChannel.....	807
can_tp1_channel_all_wakeup.....	823
can_tp1_channel_init.....	818

can_tp1_channel_all_wakeup

Syntax

```
Int32 can_tp1_channel_all_wakeup(
    const can_tp1_canChannel* canCh);
```

Include file

CanTp1.h

Purpose

To reactivate all messages that were deactivated by calling the functions `can_tp1_channel_all_sleep` and `can_tp1_msg_sleep`.

Description

Use the returned handle from the function `can_tp1_channel_init` to call this function.

Parameters

canCh Specifies the pointer to the `can_tp1_canChannel` structure.

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function has been performed without error.
CAN_TP1_BUFFER_OVERFLOW	The communication buffer occurred. Repeat the function until it returns CAN_TP1_NO_ERROR.

Example

```
can_tp1_channel_all_wakeup(canCh);
```

Related topics**References**

can_tp1_channel_all_sleep.....	822
can_tp1_msg_sleep.....	860

can_tp1_channel_BOff_go

Syntax

```
Int32 can_tp1_channel_BOff_go(
    const can_tp1_canChannel* canCh);
```

Include file

CanTp1.h

Purpose

To set the CAN channel to the bus off state. All bus operations performed by the CAN channel are canceled.

Description

Use the returned handle from the function `can_tp1_channel_init` to call this function.

Parameters

canCh Specifies the pointer to the `can_tp1_canChannel` structure.

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.

Example

```
can_tp1_channel_BOff_go(canCh);
```

Related topics

References

can_tp1_channel_BOff_return.....	825
can_tp1_channel_init.....	818

can_tp1_channel_BOff_return

Syntax

```
Int32 can_tp1_channel_BOff_return(
    const can_tp1_canChannel* canCh);
```

Include file

CanTp1.h

Purpose

To reset the slave can_tp1 CAN channel from the bus off state.

Use the returned handle from the function `can_tp1_channel_init` to call this function.

Parameters

`canCh` Specifies the pointer to the `can_tp1_canChannel` structure.

Return value

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns CAN_TP1_NO_ERROR.

Example

```
can_tp1_channel_BOff_return(canCh);
```

Related topics**References**

can_tp1_channel_BOff_go.....	824
can_tp1_channel_init.....	818

can_tp1_channel_set

Syntax

```
Int32 can_tp1_channel_set(
    const can_tp1_canChannel* canCh,
    const UInt32 mask_type,
    const UInt32 mask_value);
```

Include file

CanTp1.h

Purpose

To set a mask value or attribute for the specified CAN channel. Use this function to write the value to the specified CAN controller memory area. Use the returned handle from the function `can_tp1_channel_init` to call this function.

Parameters

canCh Specifies the pointer to the `can_tp1_canChannel` structure.

mask_type Specifies the mask type. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_CHANNEL_SET_MASK15	Sets the Message 15 Mask Register.
CAN_TP1_CHANNEL_SET_ARBMASK15	Sets the Arbitration Register for mailbox 15.
CAN_TP1_CHANNEL_SET_BAUDRATE	Sets the baud rate of the selected channel during run time.

mask_value Specifies the value of the mask to be written: 0 = "don't care", 1 = "must match".

mask_type	mask_value
CAN_TP1_CHANNEL_SET_ARBMASK15	Arbitration field for mailbox 15. Bit0 (on the right in mask_value) corresponds to bit ID0 in the arbitration field, Bit1 = ID1, ..., Bit28 = ID28.
CAN_TP1_CHANNEL_SET_MASK15	For mailbox 15 only: Message 15 Mask Register. Bit0 (on the right in mask_value) corresponds to bit ID0 in the arbitration field, Bit1 = ID1, ..., Bit28 = ID28.

mask_type	mask_value
CAN_TP1_CHANNEL_SET_BAUDRATE	Sets the baud rate (in baud). Valid range: 10,000 ... 1,000,000. Some baud rates in the allowed range cannot be met. If the actual baud rate differs from the one you specify by more than 1%, the function outputs a warning with the actual baud rate settings. Using CAN service functions, you can check the current bus status and whether the new baud rate parameters were changed correctly. Refer to CAN Service Functions on page 870.

For further information on the registers, refer to the manual of the CAN controller.

Return value This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns CAN_TP1_NO_ERROR.
CAN_TP1_BAUDRATE_L_ERROR	The baud rate is too low. The operation is aborted.
CAN_TP1_BAUDRATE_H_ERROR	The baud rate is too high. The operation is aborted.
CAN_TP1_BAUDRATE_SET_BAUDR_ERROR	Error during the calculation of the new bit timing parameters. The operation is aborted.

Messages

The following messages are defined:

Type	Message	Description
Warning	CAN_Tp1_My: baudrate on channel ... doesn't match the desired baudrate. New baudrate = ... bit/s (y: module number)	The actual baud rate differs from the one you specified by more than 1%.

Example

```
can_tp1_channel_set(
    canCh,
    CANTP1_CHANNEL_SET_MASK15,
    0xFFFFFFFF);
/* Set the Lowest bit of the Message 15 Mask Register */
/* to "don't care" */
```

Related topics

References

can_tp1_channel_init.....	818
---------------------------	-----

can_tp1_channel_txqueue_clear

Syntax

```
Int32 can_tp1_channel_txqueue_clear(
    const can_tp1_canChannel* canCh);
```

Include file

CanTp1.h

Purpose

To clear the content of the transmit queues of the selected CAN channel.

Description

The function clears the content of the transmit queues of the selected CAN channel.

Note

When you use this function, all the TX messages in the transmit queues are deleted.

Parameters

canCh Specifies the pointer to the `can_tp1_canChannel` structure.

Return value

This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	An overflow of the master to slave communication buffer occurred. Repeat the function until it returns CAN_TP1_NO_ERROR.

CAN Message Handling

Introduction

To handle different kinds of CAN messages.

Where to go from here

Information in this section

can_tp1_msg_tx_register	830
To register a transmit message on the slave can_tp1.	
can_tp1_msg_rx_register	834
To register a receive message on the slave can_tp1.	
can_tp1_msg_rqtx_register	837
To register a request transmission (RQTX) message on the slave can_tp1.	
can_tp1_msg_rqrq_register	841
To register an RQRX message on the slave can_tp1.	
can_tp1_msg_rm_register	844
To register a remote message on the slave can_tp1.	
can_tp1_msg_set	847
To set the properties of a CAN message.	
can_tp1_msg_rqtx_activate	849
To activate the request transmission message on the slave can_tp1 registered by can_tp1_msg_rqtx_register .	
can_tp1_msg_write	850
To write CAN message data.	
can_tp1_msg_send	852
To write CAN message data and send the data immediately after the delay time. To send the transmit message with new data.	
can_tp1_msg_send_id	853
To send a message with a modified identifier. This allows you to send any message ID with one registered message.	
can_tp1_msg_queue_level	855
To return the number of messages stored in the message queue allocated on the master with the can_tp1_msg_set function.	
can_tp1_msg_txqueue_init	855
To initialize the transmit queue that is used to queue messages sent by the can_tp1_msg_send_id_queued function.	
can_tp1_msg_send_id_queued	857
To build a transmit order and transmit it in the same order as the function is called.	

can_tp1_msg_txqueue_level_read	859
To read the fill level of the transmit queue for the specified TX message on the CAN slave.	
can_tp1_msg_sleep	860
To stop the transmission of the message to the CAN bus or to stop the transmission of the message data from the slave to the master.	
can_tp1_msg_wakeup	861
To reactivate a message that was deactivated by calling the <code>can_tp1_msg_sleep</code> or <code>can_tp1_channel_all_sleep</code> function.	
can_tp1_msg_read	862
To read the data length, the data, and the status information from the dual-port memory.	
can_tp1_msg_trigger	864
To send a transmit or request message immediately after the specified delay time.	
can_tp1_msg_clear	865
To clear the following message data: data[8], datalen, timestamp, deltatime, timecount, delaytime, and processed.	
can_tp1_msg_processed_register	866
To register the processed function in the command table.	
can_tp1_msg_processed_request	867
To request the message processed information from the slave can_tp1.	
can_tp1_msg_processed_read	868
To read the message processed information from the slave can_tp1.	

can_tp1_msg_tx_register

Syntax

```
can_tp1_canMsg* can_tp1_msg_tx_register(
    const can_tp1_canChannel* canCh,
    const Int32 queue,
    const UInt32 identifier,
    const UInt32 format,
    const UInt32 inform,
    const Int32 subinterrupt,
    const Float32 start_time,
    const Float32 repetition_time,
    const Float32 timeout);
```

Include file

CanTp1.h

Purpose

To register a transmit message on the slave can_tp1.

Description

If no error occurs, the function returns a pointer to the `can_tp1_canMsg` structure.

Use the returned handle when calling one of the following functions:

- `can_tp1_msg_write` to write new data to the message
- `can_tp1_msg_read` to read the returned timestamps
- `can_tp1_msg_send` to send the message with new data
- `can_tp1_msg_trigger` to send the message
- `can_tp1_msg_sleep` to deactivate the message
- `can_tp1_msg_wakeup` to reactivate the message
- `can_tp1_msg_clear` to clear the message object data
- `can_tp1_msg_processed_register` to register the processed function
- `can_tp1_msg_processed_request` to request the processed function
- `can_tp1_msg_processed_read` to read the returned data

Note

You must call `can_tp1_msg_write` to make the message valid for the CAN channel.

Parameters

canCh Specifies the pointer to the `can_tp1_canChannel` structure.

queue Specifies the communication channel within the range 0 ... 5.

identifier Specifies the identifier of the message.

format Specifies the message format. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_STD</code>	11-bit standard format, CAN 2.0A
<code>CAN_TP1_EXT</code>	29-bit extended format, CAN 2.0B

inform Specifies the information values to be updated. You can combine the predefined symbols with the logical OR operator. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_NO_INFO</code>	Returns no information.
<code>CAN_TP1_MSG_INFO</code>	Updates the message identifier and the message format.

Predefined Symbol	Meaning
CAN_TP1_TIMECOUNT_INFO	Updates the timestamp and the <code>deltatime</code> parameters.
CAN_TP1_DELAYCOUNT_INFO	Updates the <code>delaytime</code> parameter.

subinterrupt Specifies the subinterrupt number for a received message. The valid range is 0 ... 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (CAN_TP1_SUBINT_BUFFERWARN). Do not use this number for any other interrupt.

Use the following predefined symbol to select no interrupt for the TX message:

Predefined Symbol	Meaning
CAN_TP1_NO_SUBINT	No interrupt for the TX message

start_time Specifies the point in time of the first sending after timer start. Enter the value in seconds within the range 0 ... 420.

repetition_time Specifies the time interval for repeating the message automatically. Enter the value in seconds within the range 0 ... 100.

Use the following predefined symbol to define a message sent only once with `can_tp1_msg_trigger`:

Predefined Symbol	Meaning
CAN_TP1_TRIGGER_MSG	Calls <code>can_tp1_msg_trigger</code> to send the message.

timeout The message will occupy the mailbox only up to this point in time. When the threshold is exceeded, the message is released from the mailbox. Enter the value in seconds within the range 0 ... 100.

Use the following predefined symbol to calculate the timeout value internally:

Predefined Symbol	Meaning
CAN_TP1_TIMEOUT_NORMAL	The timeout value is calculated internally when registering the message. This timeout value works in most cases.

Return value

canMsg Specifies the pointer to the `can_tp1_canMsg` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	can_tp1_msg_tx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.

ID	Type	Message	Description
102	Error	can_tp1_msg_tx_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	can_tp1_msg_tx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to CAN_TP1_AUTO_INDEX.
104	Error	can_tp1_msg_tx_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	can_tp1_msg_tx_register(x,..) slave: not responding	The slave did not finish the initialization within one second.
107	Error	can_tp1_msg_tx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	can_tp1_msg_tx_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data to a filled queue. To prevent this error, deactivate all messages with <code>can_tp1_msg_sleep</code> or <code>can_tp1_channel_all_sleep</code> when registering messages or services.
140	Error	can_tp1_msg_tx_register(x,..) format: wrong format	Only the symbols CAN_TP1_STD and CAN_TP1_EXT are allowed for the parameter <code>format</code> .
141	Error	can_tp1_msg_tx_register(x,..) subint: use range 0..14!	The subinterrupt number must be within the range 0 ... 14.
142	Error	can_tp1_msg_tx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	can_tp1_msg_tx_register(x,..) id: Illegal id or id conflict	The CAN controller does not install the identifier given in the program. For further information, refer to can_tp1_canService on page 809.
144	Error	can_tp1_msg_tx_register(x,..) Too much messages (max. 100)!	The total number of registered messages is limited to 100. The program is aborted.
145	Error	can_tp1_msg_tx_register(x,..) starttime: too high (max. 420s)!	The <code>start_time</code> value must not be higher than 420 seconds. Exceeding this value causes an error and the program is aborted.
146	Error	can_tp1_msg_tx_register(x,..) rep. time: too high (max. 100s)!	The <code>repetition_time</code> value must not be higher than 100 seconds. Exceeding this value causes an error and the program is aborted.
147	Error	can_tp1_msg_tx_register(x,..) rep. time: too low !	Must be at least CAN_FRAME_TIME. A lower value causes an error and the program is aborted. Note that CAN_FRAME_TIME = (136 / Baud rate).
148	Error	can_tp1_msg_tx_register(x,..) timeout: too high (max. 100s)!	The <code>timeout</code> value must not be higher than 100 seconds. Exceeding this value causes an error and the program is aborted.
149	Error	can_tp1_msg_tx_register(x,..) timeout: too low !	The <code>timeout</code> value has to be at least 3 · CAN_FRAME_TIME. A lower value causes an error and the program is aborted. Note that CAN_FRAME_TIME = (136 / Baud rate).
152	Error	can_tp1_msg_tx_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: <ul style="list-style-type: none"> ▪ You try to register a CAN message on an uninitialized CAN channel.

ID	Type	Message	Description
			<ul style="list-style-type: none"> ▪ You try to register a CAN service on an uninitialized CAN channel. <p>Use <code>can_tp1_channel_init</code> to initialize the CAN channel.</p>

Example

For examples of how to use this function, refer to [Example of Handling Transmit and Receive Messages](#) on page 880 and [Example of Using Subinterrupts](#) on page 883.

Related topics**References**

<code>can_tp1_canMsg</code>	812
<code>can_tp1_channel_all_sleep</code>	822
<code>can_tp1_channel_init</code>	818
<code>can_tp1_msg_clear</code>	865
<code>can_tp1_msg_processed_read</code>	868
<code>can_tp1_msg_processed_register</code>	866
<code>can_tp1_msg_processed_request</code>	867
<code>can_tp1_msg_read</code>	862
<code>can_tp1_msg_send</code>	852
<code>can_tp1_msg_sleep</code>	860
<code>can_tp1_msg_trigger</code>	864
<code>can_tp1_msg_wakeup</code>	861
<code>can_tp1_msg_write</code>	850

[can_tp1_msg_rx_register](#)**Syntax**

```
can_tp1_canMsg* can_tp1_msg_rx_register(
    const can_tp1_canChannel* canCh,
    const Int32 queue,
    const UInt32 identifier,
    const UInt32 format,
    const UInt32 inform,
    const Int32 subinterrupt);
```

Include file

CanTp1.h

Purpose

To register a receive message on the slave `can_tp1`.

Description

If no error occurs, `can_tp1_msg_rx_register` returns a pointer to the `can_tp1_canMsg` structure.

Use the returned handle when calling one of the following functions:

- `can_tp1_msg_read` to read the returned data and timestamps
- `can_tp1_msg_sleep` to deactivate the message
- `can_tp1_msg_wakeup` to reactivate the message
- `can_tp1_msg_clear` to clear the message data
- `can_tp1_msg_processed_register` to register the processed function
- `can_tp1_msg_processed_request` to request the processed function
- `can_tp1_msg_processed_read` to read the returned data

Parameters

canCh Specifies the pointer to the `can_tp1_canChannel` structure.
queue Specifies the communication channel within the range 0 ... 5.
identifier Specifies the identifier of the message.
format Specifies the message format. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_STD</code>	11-bit standard format, CAN 2.0A
<code>CAN_TP1_EXT</code>	29-bit extended format, CAN 2.0B

inform Specifies the information values to be updated. You can combine the predefined symbols with the logical OR operator. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_NO_INFO</code>	Returns no information.
<code>CAN_TP1_DATA_INFO</code>	Updates the data and datalen parameters (needed for receive and request (RQRX) messages).
<code>CAN_TP1_MSG_INFO</code>	Updates the message identifier and the message format.
<code>CAN_TP1_TIMECOUNT_INFO</code>	Updates the timestamp and deltatime parameters.

subinterrupt Specifies the subinterrupt number for a received message. The valid range is 0 ... 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (`CAN_TP1_SUBINT_BUFFERWARN`). Do not use this number for any other interrupt.

Use the following predefined symbol to select no interrupt for the receive message:

Predefined Symbol	Meaning
<code>CAN_TP1_NO_SUBINT</code>	No interrupt for the receive message

Return value

canMsg This function returns the pointer to the `can_tp1_canMsg` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	can_tp1_msg_rx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	can_tp1_msg_rx_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	can_tp1_msg_rx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to CAN_TP1_AUTO_INDEX.
104	Error	can_tp1_msg_rx_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	can_tp1_msg_rx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	can_tp1_msg_rx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	can_tp1_msg_rx_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data to a filled queue. To prevent this error, deactivate all messages with <code>can_tp1_msg_sleep</code> or <code>can_tp1_channel_all_sleep</code> when registering messages or services.
140	Error	can_tp1_msg_rx_register(x,..) format: wrong format	Only the symbols CAN_TP1_STD and CAN_TP1_EXT are allowed for the parameter <code>format</code> .
141	Error	can_tp1_msg_rx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range 0 ... 14.
142	Error	can_tp1_msg_rx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	can_tp1_msg_rx_register(x,..) id: Illegal id or id conflict	The CAN controller does not install the identifier given in the program. For further information, see can_tp1_canService on page 809.
144	Error	can_tp1_msg_rx_register(x,..): Too much messages (max. 100)!	The total number of registered messages is limited to 100. The program is aborted.
152	Error	can_tp1_msg_rx_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: <ul style="list-style-type: none"> ▪ You try to register a CAN message on an uninitialized CAN channel. ▪ You try to register a CAN service on an uninitialized CAN channel. Use <code>can_tp1_channel_init</code> to initialize the CAN channel.

Example

For examples of how to use this function, refer to [Example of Handling Transmit and Receive Messages](#) on page 880 and [Example of Using Subinterrupts](#) on page 883.

```
can_tp1_canMsg* rxMsg = can_tp1_msg_rx_register(
    canCh,
    0,
    0x123,
    CAN_TP1_STD,
    CAN_TP1_DATA_INFO,
    CAN_TP1_NO_SUBINT);
```

Related topics**References**

can_tp1_channel_init.....	818
can_tp1_msg_clear.....	865
can_tp1_msg_processed_read.....	868
can_tp1_msg_processed_register.....	866
can_tp1_msg_processed_request.....	867
can_tp1_msg_read.....	862
can_tp1_msg_send.....	852
can_tp1_msg_sleep.....	860
can_tp1_msg_trigger.....	864
can_tp1_msg_wakeup.....	861
can_tp1_msg_write.....	850

can_tp1_msg_rqtx_register

Syntax

```
can_tp1_canMsg* can_tp1_msg_rqtx_register(
    const can_tp1_canChannel* canCh,
    const Int32 queue,
    const UInt32 identifier,
    const UInt32 format,
    const UInt32 inform,
    const Int32 subinterrupt,
    const Float32 start_time,
    const Float32 repetition_time,
    const Float32 timeout);
```

Include file

CanTp1.h

Purpose

To register a request transmission (RQTX) message on the slave can_tp1.

Description

Use this function to register a request message. Use the function `can_tp1_msg_rqtx_register` to register a function that receives the requested data. If no error occurs, `can_tp1_msg_rqtx_register` returns a pointer to the `can_tp1_canMsg` structure.

Use the returned handle when calling one of the following functions:

Function	Description
<code>can_tp1_msg_rqtx_activate</code>	to activate the message
<code>can_tp1_msg_read</code>	To read the returned time stamps.
<code>can_tp1_msg_sleep</code>	To deactivate the message.
<code>can_tp1_msg_wakeup</code>	To reactivate the message.
<code>can_tp1_msg_trigger</code>	To send the request message.
<code>can_tp1_msg_clear</code>	To clear the message object data.
<code>can_tp1_msg_processed_register</code>	To register the processed function.
<code>can_tp1_msg_processed_request</code>	To request the processed function.
<code>can_tp1_msg_processed_read</code>	To read the returned data.

Note

You must call `can_tp1_msg_rqtx_activate` to make the message valid for the CAN channel.

Parameters

- canCh** Specifies the pointer to the `can_tp1_canChannel` structure.
- queue** Specifies the communication channel within the range 0 ... 5.
- identifier** Specifies the identifier of the message.
- format** Specifies the message format. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_STD</code>	11-bit standard format, CAN 2.0A
<code>CAN_TP1_EXT</code>	29-bit extended format, CAN 2.0B

inform Specifies the information values to be updated. You can combine the predefined symbols with the logical OR operator; the following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_NO_INFO</code>	Returns no information.
<code>CAN_TP1_MSG_INFO</code>	Updates the message identifier and the message format.

Predefined Symbol	Meaning
CAN_TP1_TIMECOUNT_INFO	Updates the timestamp and deltatime parameters.
CAN_TP1_DELAYCOUNT_INFO	Updates the delaytime parameter.

subinterrupt Specifies the subinterrupt number for a received message. The valid range is 0 ... 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (CAN_TP1_SUBINT_BUFFERWARN). Do not use this number for any other interrupt.

Use the following predefined symbol to select no interrupt for the RQTX message:

Predefined Symbol	Meaning
CAN_TP1_NO_SUBINT	No interrupt for the RQTX messages.

start_time Specifies the point in time of the first sending after timer start. Enter the value in seconds within the range 0 ... 420.

repetition_time Specifies the time interval for repeating the message automatically. Enter the value in seconds within the range 0 ... 100.

Use the following predefined symbol to define a message sent only once with `can_tp1_msg_trigger`:

Predefined Symbol	Meaning
CAN_TP1_TRIGGER_MSG	Calls <code>can_tp1_msg_trigger</code> to send the message.

timeout The message will occupy the mailbox only up to this point in time. When the threshold is exceeded, the message is released from the mailbox. Enter the value in seconds within the range 0 ... 100.

Use the following predefined symbol to calculate the timeout value internally:

Predefined Symbol	Meaning
CAN_TP1_TIMEOUT_NORMAL	The timeout value is calculated internally when registering the message. This timeout value works in most cases.

Return value

canMsg This function returns the pointer to the `can_tp1_canMsg` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	can_tp1_msg_rqtx_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.

ID	Type	Message	Description
102	Error	can_tp1_msg_rqtx_register(x,..) queue: Illegal communication queue	There is no communication channel with this queue number.
103	Error	can_tp1_msg_rqtx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to CAN_TP1_AUTO_INDEX.
104	Error	can_tp1_msg_rqtx_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	can_tp1_msg_rqtx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	can_tp1_msg_rqtx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	can_tp1_msg_rqtx_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data to a filled queue. To prevent this error, deactivate all messages with <code>can_tp1_msg_sleep</code> or <code>can_tp1_channel_all_sleep</code> when registering messages or services.
140	Error	can_tp1_msg_rqtx_register(x,..) format: wrong format	Only the symbols CAN_TP1_STD and CAN_TP1_EXT are allowed for the parameter format.
141	Error	can_tp1_msg_rqtx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range 0 ... 14.
142	Error	can_tp1_msg_rqtx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	can_tp1_msg_rqtx_register(x,..) id: Illegal id or id conflict	The CAN controller does not install the identifier specified in the program. For further information, refer to CAN_TP1_SERVICE_MAILBOX_ERR.
144	Error	can_tp1_msg_rqtx_register(x,..): Too much messages (max. 100)!	The total number of registered messages is limited to 100. The program is aborted.
152	Error	can_tp1_msg_rqtx_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: <ul style="list-style-type: none"> ▪ You try to register a CAN message on an uninitialized CAN channel. ▪ You try to register a CAN service on an uninitialized CAN channel. Use <code>can_tp1_channel_init</code> to initialize the CAN channel.

Example

For examples of how to use this function, refer to [Example of Handling Request and Remote Messages](#) on page 881.

```
can_tp1_canMsg* rqtxMsg = can_tp1_msg_rqtx_register(
    canCh,
    0,
    0x123,
    CAN_TP1_STD,
    CAN_TP1_TIMECOUNT_INFO,
    CAN_TP1_NO_SUBINT,
    1.5,
    0.3,
    CAN_TP1_TIMEOUT_NORMAL);
```

Related topics**References**

can_tp1_channel_all_sleep.....	822
can_tp1_msg_clear.....	865
can_tp1_msg_processed_read.....	868
can_tp1_msg_processed_register.....	866
can_tp1_msg_processed_request.....	867
can_tp1_msg_read.....	862
can_tp1_msg_rqtx_activate.....	849
can_tp1_msg_send.....	852
can_tp1_msg_sleep.....	860
can_tp1_msg_trigger.....	864
can_tp1_msg_wakeup.....	861
can_tp1_msg_write.....	850

can_tp1_msg_rqrq_register

Syntax

```
can_tp1_canMsg* can_tp1_msg_rqrq_register(
    const can_tp1_canMsg* rqtxMsg,
    const UInt32 inform,
    const Int32 subinterrupt);
```

Include file

CanTp1.h

Purpose

To register an RQRX message on the slave can_tp1.

Description

Use this message to receive the data requested with an RQTX message. If no error occurs, `can_tp1_msg_rqrq_register` returns a pointer to the `can_tp1_canMsg` structure.

Use the returned handle when calling one of the following functions:

- `can_tp1_msg_read` to read the returned data and time stamps
- `can_tp1_msg_sleep` to deactivate the message
- `can_tp1_msg_wakeup` to reactivate the message
- `can_tp1_msg_clear` to clear the message object data
- `can_tp1_msg_processed_register` to register the processed function
- `can_tp1_msg_processed_request` to request the processed function
- `can_tp1_msg_processed_read` to read the returned data

Parameters

rqtxMsg Specifies the pointer to the related RQTX message.

inform Specifies the information values to be updated. You can combine the predefined symbols with the logical OR operator. The following symbols are predefined:

Predefined Symbol	Meaning
<code>CAN_TP1_NO_INFO</code>	Returns no information.
<code>CAN_TP1_DATA_INFO</code>	Updates the data and datalen parameters (needed for receive and request (RQRX) messages).
<code>CAN_TP1_MSG_INFO</code>	Updates the message identifier and the message format.
<code>CAN_TP1_TIMECOUNT_INFO</code>	Updates the timestamp and deltatime parameters.

subinterrupt Specifies the subinterrupt number for a received message. The valid range is 0 ... 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (`CAN_TP1_SUBINT_BUFFERWARN`). Do not use this number for any other interrupt.

Use the following predefined symbol to select no interrupt for the RQRX message:

Predefined Symbol	Meaning
<code>CAN_TP1_NO_SUBINT</code>	No interrupt for the RQRX message.

Return value

canMsg Specifies the pointer to the `DSxyz_canMsg` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	<code>can_tp1_msg_rqrx_register(x,..)</code> memory allocation error on master	Memory allocation error. No free memory on the master.

ID	Type	Message	Description
102	Error	can_tp1_msg_rqrx_register(x,..) queue: Illegal communication queue	There is no communication channel with this queue number.
103	Error	can_tp1_msg_rqrx_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to CAN_TP1_AUTO_INDEX.
104	Error	can_tp1_msg_rqrx_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	can_tp1_msg_rqrx_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	can_tp1_msg_rqrx_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	can_tp1_msg_rqrx_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data to a filled queue. To prevent this error, deactivate all messages with <code>can_tp1_msg_sleep</code> or <code>can_tp1_channel_all_sleep</code> when registering messages or services.
140	Error	can_tp1_msg_rqrx_register(x,..) format: wrong format	Only the symbols CAN_TP1_STD and CAN_TP1_EXT are allowed for the parameter format.
141	Error	can_tp1_msg_rqrx_register(x,..) subint: use range 0..14 !	The subinterrupt number must be within the range 0 ... 14.
142	Error	can_tp1_msg_rx_register(x,..) subint: used for busoff!	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	can_tp1_msg_rqrx_register(x,..) id: Illegal id or id conflict	The CAN controller does not install the identifier specified in the program. For further information, see can_tp1_canService on page 809.
144	Error	can_tp1_msg_rqrx_register(x,..): Too much messages (max. 100)!	The total number of registered messages is limited to 100. The program is aborted.
152	Error	can_tp1_msg_rqrx_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: <ul style="list-style-type: none"> ▪ You try to register a CAN message on an uninitialized CAN channel. ▪ You try to register a CAN service on an uninitialized CAN channel. Use <code>can_tp1_channel_init</code> to initialize the CAN channel.

Example

For examples of how to use this function, refer to [Example of Handling Request and Remote Messages](#) on page 881.

```
can_tp1_canMsg* rqrxMsg = can_tp1_msg_rqrx_register(
    rqtxMsg,
    CAN_TP1_DATA_INFO,
    CAN_TP1_NO_SUBINT);
```

Related topics**References**

can_tp1_canMsg.....	812
can_tp1_channel_all_sleep.....	822
can_tp1_channel_init.....	818
can_tp1_msg_clear.....	865
can_tp1_msg_processed_read.....	868
can_tp1_msg_processed_register.....	866
can_tp1_msg_processed_request.....	867
can_tp1_msg_read.....	862
can_tp1_msg_sleep.....	860
can_tp1_msg_wakeup.....	861

[can_tp1_msg_rm_register](#)**Syntax**

```
can_tp1_canMsg* can_tp1_msg_rm_register(
    const can_tp1_canChannel* canCh,
    const Int32 queue,
    const UInt32 identifier,
    const UInt32 format,
    const UInt32 inform,
    const Int32 subinterrupt);
```

Include file

CanTp1.h

Purpose

To register a remote message on the slave can_tp1.

Description

If no error occurs, the function returns a pointer to the `can_tp1_canMsg` structure.

Use the returned handle when calling one of the following functions:

- `can_tp1_msg_write` to support the remote message with data
- `can_tp1_msg_read` to read the returned time stamps
- `can_tp1_msg_sleep` to deactivate the message
- `can_tp1_msg_wakeup` to reactivate the message
- `can_tp1_msg_clear` to clear the message object data
- `can_tp1_msg_processed_register` to register the processed function
- `can_tp1_msg_processed_request` to request the processed function
- `can_tp1_msg_processed_read` to read the returned data

A remote message is a special kind of a transmit message. It is sent only if the CAN controller has received an associated request message and carries the requested data.

Note

A remote message permanently occupies a mailbox on the slave can_tp1 CAN channel. Therefore, only 10 remote messages are allowed within the same model for each CAN channel to ensure secure CAN operation. If this is not done, the function outputs an error and aborts the program.

Parameters

- canCh** Specifies the pointer to the `can_tp1_canChannel` structure.
- queue** Specifies the communication channel within the range 0 ... 5.
- identifier** Specifies the identifier of the message.
- format** Specifies the message format. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_STD	11-bit standard format, CAN 2.0A
CAN_TP1_EXT	29-bit extended format, CAN 2.0B

- inform** Specifies the information values to be updated. You can combine the predefined symbols with the logical OR operator. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_INFO	Returns no information.
CAN_TP1_MSG_INFO	Updates the message identifier and the message format.
CAN_TP1_TIMECOUNT_INFO	Updates the timestamp and the <code>deltatime</code> parameters.
CAN_TP1_DELAYCOUNT_INFO	Updates the <code>delaytime</code> parameter.

- subinterrupt** Specifies the subinterrupt number for a received message. The valid range is 0 ... 14.

Note

The interrupt number 15 is occupied by the buffer overflow warning interrupt (CAN_TP1_SUBINT_BUFFERWARN). You must not use this number for any other interrupt.

Use the following predefined symbol to select no interrupt for the RM message:

Predefined Symbol	Meaning
CAN_TP1_NO_SUBINT	No interrupt for the RM message

Return value

- canMsg** This function returns the pointer to the `can_tp1_canMsg` structure.

Messages

The following error and warning messages are defined:

ID	Type	Message	Description
101	Error	can_tp1_msg_rm_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	can_tp1_msg_rm_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	can_tp1_msg_rm_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to CAN_TP1_AUTO_INDEX.
104	Error	can_tp1_msg_rm_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	can_tp1_msg_rm_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	can_tp1_msg_rm_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	can_tp1_msg_rm_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data to a filled queue. To prevent this error, deactivate all messages with <code>can_tp1_msg_sleep</code> or <code>can_tp1_channel_all_sleep</code> when registering messages or services.
140	Error	can_tp1_msg_rm_register(x,..) format: wrong format	Only the symbols CAN_TP1_STD and CAN_TP1_EXT are allowed for the parameter format.
141	Error	can_tp1_msg_rm_register(x,..) subint: use range 0...14.	The subinterrupt number must be within the range 0 ... 14.
142	Error	can_tp1_msg_rm_register(x,...) subint: used for busoff.	The specified subinterrupt number is used for the CAN channel bus off subinterrupt.
143	Error	can_tp1_msg_rm_register(x,..) id: illegal id or id conflict	The CAN controller does not install the identifier specified in the program. For further information, see can_tp1_canService on page 809.
144	Error	can_tp1_msg_rm_register(x,..) Too much messages (max. 100).	The total number of registered messages is limited to 100. The program is aborted.
150	Error	can_tp1_msg_rm_register(x,...) no rm mailbox free (max. 10).	For each channel, only 10 remote messages are allowed within the same model to ensure secure CAN operation. If this is not done, the function outputs an error and the program is aborted.
152	Error	can_tp1_msg_rm_register(x,...) canCh: the CAN channel wasn't initialized	This message is displayed if: <ul style="list-style-type: none"> ▪ You try to register a CAN message on an uninitialized CAN channel. ▪ You try to register a CAN service on an uninitialized CAN channel. Use <code>can_tp1_channel_init</code> to initialize the CAN channel.

Example

For examples of how to use this function, refer to [Example of Handling Request and Remote Messages](#) on page 881.

```
can_tp1_canMsg* rmMsg = can_tp1_msg_rm_register(
    canCh,
    0,
    0x123,
    CAN_TP1_STD,
    CAN_TP1_TIMECOUNT_INFO,
    CAN_TP1_NO_SUBINT);
```

Related topics**References**

can_tp1_canMsg.....	812
can_tp1_msg_clear.....	865
can_tp1_msg_processed_read.....	868
can_tp1_msg_processed_register.....	866
can_tp1_msg_processed_request.....	867
can_tp1_msg_read.....	862
can_tp1_msg_sleep.....	860
can_tp1_msg_trigger.....	864
can_tp1_msg_wakeup.....	861
can_tp1_msg_write.....	850

can_tp1_msg_set

Syntax

```
Int32 can_tp1_msg_set(
    can_tp1Msg* msg,
    const UInt32 type,
    const void* value );
```

Include file

CanTp1.h

Purpose

To set the properties of a CAN message.

Description

This function allows you to

- Receive different message IDs with one message via a bitmask (type = DS1202_CAN_MSG_MASK),
- Set the send period for a TX or RQ message (type = DS1202_CAN_MSG_PERIOD),
- Set the identifier for a TX or RQ message (type = DS1202_CAN_MSG_ID) or

- Set the queue depth for a message (type = DS1202_CAN_MSG_QUEUE).
- Set the length for a message (type = DS1202_CAN_MSG_LEN).

Note

For DS1202_CAN_MSG_MASK the following rules apply:

- For each CAN channel, only one mask for STD and one mask for EXT messages is allowed.
- If you call `can_tp1_msg_set` for another message, the bitmask is removed from the first message.
- Using the bitmask might cause conflicts with messages installed for one message ID. In this case, message data is received via the message installed for this ID.
- You can skip the bitmask by setting all bits to "must match" (0xFFFFFFFF) again.

Parameters

msg Specifies the pointer to the message structure.

type Defines the property to be specified. Use one of the predefined symbols:

Predefined Symbol	Meaning
CAN_TP1_MSG_MASK	To set the arbitrary mask for an RX message
CAN_TP1_MSG_PERIOD	To set the send period for a TX or RQ message
CAN_TP1_MSG_ID	To set the identifier for a TX or RQ message
CAN_TP1_MSG_QUEUE	To set the queue depth for a message
CAN_TP1_MSG_LEN	To set the data length code (DLC) for a TX, RQTX, or RM message

value Specifies the value to be set for the defined **type**.

For the CAN_TP1_MSG_LEN type, you can specify the data length code (DLC) value (UInt32) in the range 0 ... 8 bytes.

Note

If the specified length exceeds 8 bytes, the function sets the length to 8 bytes.

Return value

This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.
CAN_TP1_TYPE_ERROR	The operation is not allowed for the specified message object.

Predefined Symbol	Meaning
CAN_TP1_MSG_TYPE_ERROR	The function is not available for the specified message type. It is available only for TX, RQTX, and RM messages.

Example

This example shows how to receive different message IDs with one message:

Install one message with a bitmask that allows you to set some bits of the mask to "don't care" via `can_tp1_msg_set`.

```
UInt32 mask = 0xFFFFFFFF0; // Sets the last four bits to
                           // "Don't Care".
can_tp1_msg_set( msg, CAN_TP1_MSG_MASK, &mask );
```

Example

This example shows how to receive different message IDs with one message via a bitmask:

- A message with ID 0x120 was registered. Now, you set the bitmask via `can_tp1_msg_set(msg, CAN_TP1_MSG_MASK, &mask);` with `mask = 0xFFFFFFFF0`.

This lets you receive the message IDs 0x120, 0x121, ..., 0x12F.

- A message with ID 0x120 was registered. Now, you set the bitmask to 0x1FFFFFFF. This lets you receive the message IDs 0x120 and 0x130.

Related topics**References**

<code>can_tp1_msg_read</code>	862
-------------------------------------	-----

can_tp1_msg_rqtx_activate

Syntax

```
Int32 can_tp1_msg_rqtx_activate(
    const can_tp1_canMsg* canMsg);
```

Include file

CanTp1.h

Purpose

To activate the request transmission message on the slave `can_tp1` registered by `can_tp1_msg_rqtx_register`.

Description

This function does not send the message. Sending the message is done by the timer for cyclic sending or by calling `can_tp1_msg_trigger` for acyclic sending.

Use the returned handle from the function `can_tp1_msg_rqtx_register` to call this function.

Parameters	<code>canMsg</code> Specifies the pointer to the <code>can_tp1_canMsg</code> structure.
-------------------	---

Return value	This function returns the error code. The following symbols are predefined:
---------------------	---

Predefined Symbol	Meaning
<code>CAN_TP1_NO_ERROR</code>	The function was performed without error.
<code>CAN_TP1_BUFFER_OVERFLOW</code>	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns <code>CAN_TP1_NO_ERROR</code> .
<code>CAN_TP1_TYPE_ERROR</code>	The operation is not allowed for the given message object.

Example	For examples of how to use this function, refer to Example of Handling Request and Remote Messages on page 881.
----------------	---

Related topics	References				
	<table border="0"> <tr> <td><code>can_tp1_msg_rqtx_register.....</code></td> <td>837</td> </tr> <tr> <td><code>can_tp1_msg_trigger.....</code></td> <td>864</td> </tr> </table>	<code>can_tp1_msg_rqtx_register.....</code>	837	<code>can_tp1_msg_trigger.....</code>	864
<code>can_tp1_msg_rqtx_register.....</code>	837				
<code>can_tp1_msg_trigger.....</code>	864				

can_tp1_msg_write

Syntax	<pre>Int32 can_tp1_msg_write(const can_tp1_canMsg* canMsg, const UInt32 datalen, const UInt32* data);</pre>
---------------	--

Include file	<code>CanTp1.h</code>
---------------------	-----------------------

Purpose	To write CAN message data.
----------------	----------------------------

Description	<p>There are differences for the following message types:</p> <ul style="list-style-type: none"> ▪ TX message <p>Calling this function for the first time prepares the message to be sent with the specified parameters in the message register function. A TX message with a repetition time is sent automatically with the specified value. A TX message registered by CAN_TP1_TRIGGER_MSG is sent only when calling <code>can_tp1_msg_trigger</code> or <code>can_tp1_msg_send</code>.</p> <p>Calling this function again updates CAN message data and data length.</p> <ul style="list-style-type: none"> ▪ RM message <p>Calling this function for the first time prepares and activates the remote message to be sent with the specified data and data length. The remote message is sent when a corresponding request message is received.</p> <p>Calling this function again updates CAN message data and data length.</p> <p>Use the returned handle from the function <code>can_tp1_msg_tx_register</code> or <code>can_tp1_msg_rm_register</code> to call this function.</p>
Parameters	<p>canMsg Specifies the pointer to the <code>can_tp1_canMsg</code> structure.</p> <p>datalen Specifies the length of the CAN message data. The valid range is 0 ... 8 bytes.</p> <p>data Specifies the buffer for CAN message data.</p>

Return value	This function returns the error code; the following symbols are predefined:
---------------------	---

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function has been performed without error.
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.
CAN_TP1_TYPE_ERROR	The operation is not allowed for the specified message object.

Example	For examples, refer to:
	<ul style="list-style-type: none"> ▪ Example of Handling Transmit and Receive Messages on page 880 ▪ Example of Handling Request and Remote Messages on page 881 ▪ Example of Using Subinterrupts on page 883

Related topics	References				
	<table border="0"> <tr> <td><code>can_tp1_msg_rm_register</code>.....</td> <td>844</td> </tr> <tr> <td><code>can_tp1_msg_send</code>.....</td> <td>852</td> </tr> </table>	<code>can_tp1_msg_rm_register</code>	844	<code>can_tp1_msg_send</code>	852
<code>can_tp1_msg_rm_register</code>	844				
<code>can_tp1_msg_send</code>	852				

can_tp1_msg_trigger.....	864
can_tp1_msg_tx_register.....	830

can_tp1_msg_send

Syntax

```
Int32 can_tp1_msg_send(
    const can_tp1_canMsg* canMsg,
    const UInt32 datalen,
    const UInt32* data,
    const Float32 delay);
```

Include file

CanTp1.h

Purpose

To write CAN message data and send the data immediately after the delay time.
To send the transmit message with new data.

Description

The transmit message must have been registered by calling `can_tp1_msg_tx_register`. Then `can_tp1_msg_send` writes the CAN message data to the dual-port memory. After this, the message is set up on the CAN controller and the sending of the message is started. The message is sent according to the specified parameters in the register function.

Use the returned handle from the function `can_tp1_msg_tx_register` to call this function.

Note

Suppose the `can_tp1_msg_send` function is called twice. If the interval between the function calls is short, the second function call might occur before the TX message was sent by the first function call. In this case, the TX message is sent only once, with the data of the second function call.

Parameters

- canMsg** Specifies the pointer to the `can_tp1_canMsg` structure.
- datalen** Specifies the length of the CAN message data. The valid range is 0 ... 8 bytes.
- data** Specifies the buffer for CAN message data.
- delay** Sends the message after the delay time. The valid range is 0.0 ... 100.0 seconds.

Return value This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.
CAN_TP1_TYPE_ERROR	The operation is not allowed for the specified message object.

Example

```
UInt32 txData[8] = {1,2,3,4,5,6,7,8};
can_tp1_msg_send (txMsg, 8, txData, 0.005);
```

Related topics**References**

can_tp1_msg_send_id.....	853
can_tp1_msg_tx_register.....	830

[can_tp1_msg_send_id](#)**Syntax**

```
Int32 can_tp1_msg_send_id (
    can_tp1Msg* canMsg,
    const UInt32 id,
    const UInt32 datalen,
    const UInt8* data,
    const Float32 delay);
```

Include file

CanTp1.h

Purpose

To send a message with a modified identifier. This lets you send any message ID with one registered message.

Parameters	<p>canMsg Specifies the pointer to the <code>can_tp1_canMsg</code> structure.</p> <p>id Specifies the ID of the message to be modified.</p> <p>datalen Specifies the length of the CAN message data. The valid range is 0 ... 8 bytes.</p> <p>data Specifies the buffer for CAN message data.</p> <p>delay Sends the message after the delay time. The valid range is 0.0 ... 100.0 seconds.</p>
-------------------	---

Return value	This function returns the error code. The following symbols are predefined:
---------------------	---

Predefined Symbol	Meaning
<code>CAN_TP1_NO_ERROR</code>	The function was performed without error.
<code>CAN_TP1_BUFFER_OVERFLOW</code>	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns <code>CAN_TP1_NO_ERROR</code> .
<code>CAN_TP1_TYPE_ERROR</code>	The operation is not allowed for the specified message object.

Note

- The message format is determined by the format in which the message was installed when it was used for the first time.
 - You have to use a handshake mechanism to send a message via `can_tp1_msg_send_id` to make sure that a message installed for the message object has been sent already.
- Each message object is buffered only once on the slave. This might cause conflicts when you try to send several message objects with different IDs.

Example

The `can_tp1_msg_send_id` function lets you send any message ID with one registered message.

```
can_tp1_msg_send_id(msg, 0x123, data, 8, 0.001)
```

Related topics**References**

<code>can_tp1_msg_queue_level</code>	855
<code>can_tp1_msg_send</code>	852
<code>can_tp1_msg_tx_register</code>	830

can_tp1_msg_queue_level

Syntax

```
Int32 can_tp1_msg_queue_level (
    can_tp1_canMsg* canMsg);
```

Include file

CanTp1.h

Purpose

To return the number of messages stored in the message queue allocated on the master with `can_tp1_msg_set(msg, CAN_TP1_MSG_QUEUE, &size)`.

Description

Use `can_tp1_msg_read` to copy the messages from the communication channel to the message buffer.

Note

This is not the number of messages in the DPMEM.

Parameters

`canMsg` Specifies the pointer to the `can_tp1_canMsg` structure.

Return value

This function returns the number of messages in the message queue.

Related topics
References

can_tp1_msg_read.....	862
can_tp1_msg_set.....	847

can_tp1_msg_txqueue_init

Syntax

```
Int32 can_tp1_msg_txqueue_init(
    can_tp1Msg* canMsg,
    const UInt32 overrun_policy,
    Float32 delay);
```

Include file

CanTp1.h

Purpose	To initialize the transmit queue that is used to queue messages sent by the <code>can_tp1_msg_send_id_queued</code> function.						
Description	The function allocates a circular buffer on the slave with the specified overrun policy, where the transmit orders from the <code>can_tp1_msg_send_id_queued</code> function are stored. The queue stores up to 64 message entries.						
Parameter	<p>canMsg Specifies the pointer to the <code>can_tp1_canMsg</code> structure.</p> <p>overrun_policy Selects the overrun policy of the transmit queue. The following symbols are predefined:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>CAN_TP1_TXQUEUE_OVERRUN_OVERWRITE</code></td><td>The oldest message is overwritten.</td></tr> <tr> <td><code>CAN_TP1_TXQUEUE_OVERRUN_IGNORE</code></td><td>The oldest message is kept. The new message is lost.</td></tr> </tbody> </table> <p>delay Specifies the delay between the messages of the transmit queue within the range 0.0 ... 10 s.</p>	Predefined Symbol	Meaning	<code>CAN_TP1_TXQUEUE_OVERRUN_OVERWRITE</code>	The oldest message is overwritten.	<code>CAN_TP1_TXQUEUE_OVERRUN_IGNORE</code>	The oldest message is kept. The new message is lost.
Predefined Symbol	Meaning						
<code>CAN_TP1_TXQUEUE_OVERRUN_OVERWRITE</code>	The oldest message is overwritten.						
<code>CAN_TP1_TXQUEUE_OVERRUN_IGNORE</code>	The oldest message is kept. The new message is lost.						
<p>Note</p> <ul style="list-style-type: none"> Even if a delay of 0 seconds is specified, the distance between two message frames is greater than 0. The length of this gap depends on the load of the slave. If the delay is smaller than 0, the function sets the delay to 0. The real delay between two message frames might not be constant due to jitter. The jitter of the delay also depends on the load of the slave. Only two message objects (one STD and one EXT format message) can be used for queuing for every channel. Nevertheless <code>can_tp1_msg_send_id_queued</code> allows the identifier of the message object to be changed. The function can be called again to change the delay or to assign the transmit queue to another message. The old messages in the transmit queue are lost (not transmitted) if the transmit queue is initialized again. 							
Return value	This function returns the error code. The following symbols are predefined:						
Predefined Symbol	Meaning						
<code>CAN_TP1_NO_ERROR</code>	The transmit queue was initialized successfully.						
<code>CAN_TP1_BUFFER_OVERFLOW</code>	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns <code>CAN_TP1_NO_ERROR</code> .						
<code>CAN_TP1_TXQUEUE_INIT_NOT_REG_ERROR</code>	The message (<code>canMsg</code>) was not registered. The operation is aborted.						
<code>CAN_TP1_TXQUEUE_INIT_MSG_TYPE_ERROR</code>	The message (<code>canMsg</code>) is not a TX message. The operation is aborted.						

Messages

The following messages are defined:

ID	Type	Message	Description
154	Error	can_tp1_msg_txqueue_init(): TX message is not registered	The message was not registered successfully.
155	Error	can_tp1_msg_txqueue_init(): not a TX message	The specified message is not a TX message.
301	Warning	can_tp1_msg_txqueue_init(): delay time: too high (max. 10 s). Set to maximum.	The delay time must be within the range 0 ... 10 s.

Example

The following example shows you how to initialize a TX queue.

```
void main()
{
    can_tp1_canMsg* txMsg;
    ...
    txMsg = can_tp1_msg_tx_register( txCh,
                                    2, 0x1, CAN_TP1_STD,
                                    CAN_TP1_TIMECOUNT_INFO |
                                    CAN_TP1_MSG_INFO,
                                    1, 0.0,
                                    CAN_TP1_TRIGGER_MSG, 0 );
    can_tp1_msg_txqueue_init (
        txMsg, CAN_TP1_TXQUEUE_OVERRUN_OVERWRITE, 0.01 );
    ...
}
```

Related topics**References**

[can_tp1_msg_send_id_queued](#)..... 857

can_tp1_msg_send_id_queued

Syntax

```
Int32 can_tp1_msg_send_id_queued(
    can_tp1Msg* canMsg,
    const UInt32 id,
    const UInt32 data_len,
    const UInt32* data);
```

Include file

CanTp1.h

Purpose	To build a transmit order and transmit it in the same order as the function is called.							
Description	<p>If no queue overflow occurs, each message is transmitted. In the case of queue overflow (number of messages is greater than 64), the newest message overwrites the oldest one or the oldest messages are kept while new messages are lost. See can_tp1_msg_txqueue_init on page 855.</p> <p>The CAN_TP1_SERVICE_TXQUEUE_OVERFLOW_COUNT service allows the overflow counter of the transmit queue to be requested to check whether an overflow occurred.</p>							
Parameter	<p>canMsg Specifies the pointer to the can_tp1_canMsg structure.</p> <p>id Specifies the CAN message identifier type (STD/EXT). The identifier type must correspond to the type (STD/EXT) of the registered message object. This allows the identifier of the message object to be changed during run time.</p> <p>data_len Specifies the length of data within the range 0 ... 8.</p> <p>data Specifies the message data.</p>							
Return value	This function returns the error code. The following symbols are predefined:							
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Predefined Symbol</th> <th style="text-align: left; padding: 2px;">Meaning</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">CAN_TP1_NO_ERROR</td> <td style="padding: 2px;">The transmit queue was initialized successfully.</td> </tr> <tr> <td style="padding: 2px;">CAN_TP1_BUFFER_OVERFLOW</td> <td style="padding: 2px;">Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.</td> </tr> <tr> <td style="padding: 2px;">CAN_TP1_SEND_ID_QUEUED_INIT_ERROR</td> <td style="padding: 2px;">The transmit queue for TX messages was not initialized.</td> </tr> </tbody> </table>	Predefined Symbol	Meaning	CAN_TP1_NO_ERROR	The transmit queue was initialized successfully.	CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.	CAN_TP1_SEND_ID_QUEUED_INIT_ERROR	The transmit queue for TX messages was not initialized.
Predefined Symbol	Meaning							
CAN_TP1_NO_ERROR	The transmit queue was initialized successfully.							
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.							
CAN_TP1_SEND_ID_QUEUED_INIT_ERROR	The transmit queue for TX messages was not initialized.							
Messages	The following messages are defined:							
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">ID</th> <th style="text-align: left; padding: 2px;">Type</th> <th style="text-align: left; padding: 2px;">Message</th> <th style="text-align: left; padding: 2px;">Description</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">153</td> <td style="padding: 2px;">Error</td> <td style="padding: 2px;">can_tp1_msg_send_id_queued(): TX queue: Not initialized!</td> <td style="padding: 2px;">The transmit queue was not initialized.</td> </tr> </tbody> </table>	ID	Type	Message	Description	153	Error	can_tp1_msg_send_id_queued(): TX queue: Not initialized!	The transmit queue was not initialized.
ID	Type	Message	Description					
153	Error	can_tp1_msg_send_id_queued(): TX queue: Not initialized!	The transmit queue was not initialized.					
Example	The following example shows how to build a transmit sequence for a TX queue.							
<pre style="font-family: monospace; margin: 0;">void main() { can_tp1_canMsg* txMsg; UInt32 txMsgData[8]; ... }</pre>								

```

txMsg = can_tp1_msg_tx_register( txCh,
                                2, 0x1, CAN_TP1_STD,
                                CAN_TP1_TIMECOUNT_INFO |
                                CAN_TP1_MSG_INFO,
                                1, 0.0,
                                CAN_TP1_TRIGGER_MSG, 0 );
/* initialize a transmit queue with delay = 0.01 s */
can_tp1_msg_txqueue_init (
    txMsg, CAN_TP1_TXQUEUE_OVERRUN_OVERWRITE; 0.01);
...
/* Write three messages to the transmit queue. */
/*The first message is transmitted immediately. */
/*The following messages are transmitted with a */
/*timely distance of 0.01 s. */
txMsgData[0] = 0x01;
can_tp1_msg_send_id_queued(txMsg, 0x12, 1, txMsgData);
txMsgData[0] = 0x02;
can_tp1_msg_send_id_queued(txMsg, 0x13, 1, txMsgData);
txMsgData[0] = 0x03;
can_tp1_msg_send_id_queued(txMsg, 0x14, 1, txMsgData);
...
}

```

Related topics

References

can_tp1_msg_txqueue_init.....	855
-------------------------------	-----

can_tp1_msg_txqueue_level_read

Syntax

```
UInt32 can_tp1_msg_txqueue_level_read(
    const can_tp1Msg* canMsg);
```

Include file

CanTp1.h

Purpose

To read the fill level of the transmit queue for the specified TX message on the CAN slave.

Description	The function reads the fill level of the transmit queue for the specified TX message on the CAN slave.
--------------------	--

Note

The TX messages pending in the command queue between the CAN master and the CAN slave are not taken into account.

Parameter	canMsg Specifies the pointer to the <code>can_tp1_canMsg</code> structure.
Return value	Level of TX-queue The number of TX messages in the transmit queue on the CAN slave (0 ... 64).

can_tp1_msg_sleep

Syntax	<pre>Int32 can_tp1_msg_sleep(const can_tp1_canMsg* canMsg);</pre>
Include file	<code>CanTp1.h</code>
Purpose	<p>The purpose depends on the message type:</p> <ul style="list-style-type: none"> ▪ TX, RQTX, and RM messages To stop the transmission of the message to the CAN bus. ▪ RX and RQRX messages To stop the transmission of the message data from the slave to the master.
Description	The message is deactivated and remains in sleep mode until it is reactivated by calling <code>can_tp1_msg_wakeup</code> or <code>can_tp1_channel_all_wakeup</code> .
Parameters	canMsg Specifies the pointer to the <code>can_tp1_canMsg</code> structure.

Return value This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.
CAN_TP1_TYPE_ERROR	The operation is not allowed for the given message object.

Example

```
can_tp1_msg_sleep(txMsg);
```

Related topics**References**

can_tp1_channel_all_sleep.....	822
can_tp1_channel_all_wakeup.....	823
can_tp1_msg_wakeup.....	861

can_tp1_msg_wakeup

Syntax

```
Int32 can_tp1_msg_wakeup(
    const can_tp1_canMsg* canMsg);
```

Include file

CanTp1.h

Purpose

To reactivate a message that has been deactivated by calling the `can_tp1_msg_sleep` or `can_tp1_channel_all_sleep` function.

Parameters

canMsg Specifies the pointer to the `can_tp1_canMsg` structure.

Return value This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.
CAN_TP1_TYPE_ERROR	The operation is not allowed for the given message object.

Example

```
can_tp1_msg_wakeup(txMsg);
```

Related topics**References**

can_tp1_channel_all_sleep.....	822
can_tp1_channel_all_wakeup.....	823
can_tp1_msg_sleep.....	860

can_tp1_msg_read

Syntax

```
Int32 can_tp1_msg_read(
    can_tp1_canMsg* canMsg);
```

Include file

CanTp1.h

Purpose

To read the data length, the data, and the status information from the dual-port memory.

Description

The return value provides information on whether or not the data is new. If not, the existing parameter values remain unchanged.

You can call this function several times for one message object to read all the messages available in the message buffer (see also [can_tp1_msg_set](#) on page 847). By default, only one message can be received.

Use the function `can_tp1_msg_clear` to clear the message data and time stamps. This is useful for simulation start/stop transitions.

Note

The status information that is returned depends on the previously specified inform parameter in the register function that corresponds to the message.

Parameters

`canMsg` Specifies the pointer to the `can_tp1_canMsg` structure.

Parameter	Meaning
<code>data</code>	Buffer with the updated data
<code>datalen</code>	Data length of the message
<code>deltatetime</code>	Delta time of the message
<code>timestamp</code>	Time stamp of the message
<code>delaytime</code>	Delaytime of the message
<code>processed</code>	Processed flag of the message
<code>identifier</code>	Identifier of the message
<code>format</code>	Format of the identifier

Return value

This function returns the error code. The following symbols are predefined:

Symbols	Meaning
<code>CAN_TP1_NO_ERROR</code>	The function was performed without error.
<code>CAN_TP1_NO_DATA</code>	No data was updated.
<code>CAN_TP1_DATA_LOST</code>	The input data of a previous request for the specified function was overwritten.

Example

For examples, refer to:

- [Example of Handling Transmit and Receive Messages](#) on page 880
- [Example of Handling Request and Remote Messages](#) on page 881
- [Example of Using Subinterrupts](#) on page 883

Related topics

References

<code>can_tp1_msg_clear</code>	865
<code>can_tp1_msg_set</code>	847

can_tp1_msg_trigger

Syntax

```
Int32 can_tp1_msg_trigger(
    const can_tp1_canMsg* canMsg,
    const Float32 delay);
```

Include file

CanTp1.h

Purpose

To send a transmit or request message immediately after the specified delay time.

Description

This function can be used for acyclic message sending. Use the returned handle from the `can_tp1_msg_tx_register` or `can_tp1_msg_rqtx_register` function to call this function.

Parameters

canMsg Specifies the pointer to the `can_tp1_canMsg` structure.

delay Sends the message after the delay time. The valid range is 0.0 ... 100.0 seconds.

Return value

This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.
CAN_TP1_TYPE_ERROR	The operation is not allowed for the specified message object.

Example

```
can_tp1_msg_trigger(txMsg, 0.005); /* 5 ms delay */
```

Related topics

References

can_tp1_msg_rqtx_register.....	837
can_tp1_msg_tx_register.....	830

can_tp1_msg_clear

Syntax

```
void can_tp1_msg_clear(  
    can_tp1_canMsg* canMsg);
```

Include file

CanTp1.h

Purpose

To clear the following message data: data[8], datalen, timestamp, deltatime, timecount, delaytime and processed.

Description

This is useful for simulation start/stop transitions.

Use the returned handle from the message register functions to call this function.

Note

The structure members identifier, format, module, queue, index, msg_no, type, inform, canChannel, and msgService are untouched, because any manipulation of these structure members would corrupt the message object.

Parameters

canMsg Specifies the pointer to the `can_tp1_canMsg` structure.

Return value

None

Example

```
can_tp1_msg_clear(rxMsg);
```

Related topics**References**

can_tp1_all_data_clear.....	877
can_tp1_msg_rm_register.....	844
can_tp1_msg_rqrx_register.....	841
can_tp1_msg_rqtx_register.....	837
can_tp1_msg_rx_register.....	834
can_tp1_msg_tx_register.....	830

can_tp1_msg_processed_register

Syntax

```
void can_tp1_msg_processed_register(
    can_tp1_canMsg* canMsg);
```

Include file

CanTp1.h

Purpose

To register the processed function in the command table.

Use `can_tp1_msg_processed_read` to read the processed flag and time stamp without registering the message with the inform parameter `CAN_TP1_TIMECOUNT_INFO`.

Parameters

`canMsg` Specifies the pointer to the `can_tp1_canMsg` structure.

Return value

None

Messages

The following error and warning messages are defined:

ID	Type	Description	Message
101	Error	can_tp1_msg_processed_register(x,...) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	can_tp1_msg_processed_register(x,...) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	can_tp1_msg_processed_register(x,...) index: illegal function index	The index does not exist in the command table and is not equal to <code>CAN_TP1_AUTO_INDEX</code> .
104	Error	can_tp1_msg_processed_register(x,...) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	can_tp1_msg_processed_register(x,...) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	can_tp1_msg_processed_register(x,...) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	can_tp1_msg_processed_register(x,...) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data to a filled queue. To prevent this error, deactivate all messages with <code>can_tp1_msg_sleep</code> or <code>can_tp1_channel_all_sleep</code> when registering messages or services.

Example

```
can_tp1_msg_processed_register(rxMsg);
```

Related topics**References**

can_tp1_channel_all_sleep.....	822
can_tp1_channel_init.....	818
can_tp1_msg_processed_read.....	868
can_tp1_msg_processed_request.....	867
can_tp1_msg_sleep.....	860

can_tp1_msg_processed_request

Syntax

```
Int32 can_tp1_msg_processed_request(  
    const can_tp1_canMsg* canMsg);
```

Include file

CanTp1.h

Purpose

To request the message processed information from the slave can_tp1.

Parameters

canMsg Specifies the pointer to the can_tp1_canMsg structure.

Return value

This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.
CAN_TP1_NO_DATA	can_tp1_msg_processed_request was called without registering the function with can_tp1_msg_processed_register or an empty canMsg structure was handled.

Example

```
can_tp1_msg_processed_request(rxMsg);
```

Related topics**References**

can_tp1_msg_processed_read.....	868
can_tp1_msg_processed_register.....	866

can_tp1_msg_processed_read

Syntax

```
Int32 can_tp1_msg_processed_read(
    can_tp1_canMsg* canMsg,
    Float64* timestamp,
    UInt32* processed);
```

Include file

CanTp1.h

Purpose

To read the message processed information from the slave can_tp1.

Description

Prior to this, this information must have been requested by the master calling the function `can_tp1_msg_processed_request` that demands the processed flag and the time stamp from the slave can_tp1.

Parameters

- canMsg** Specifies the pointer to the `can_tp1_canMsg` structure.
- timestamp** Specifies the time stamp when the message was last sent or received.
- processed** Specifies the processed flag of the message. The following symbols are predefined:

Symbols	Meaning
CAN_TP1_PROCESSED	Message has been sent/received since the last execution call.
CAN_TP1_NOT_PROCESSED	Message has not been sent/received since the last execution call.

Return value

This function returns the error code. The following symbols are predefined:

Symbols	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_NO_DATA	No data was updated.
CAN_TP1_DATA_LOST	The input data of a previous request for the specified function was overwritten.

Related topics**References**

can_tp1_msg_processed_register.....	866
can_tp1_msg_processed_request.....	867

CAN Service Functions

Introduction To get information on errors and status information.

Where to go from here	Information in this section
	can_tp1_service_register870 To register the service function.
	can_tp1_service_request872 To request the service information from the slave can_tp1.
	can_tp1_service_read873 To read the service information from the slave can_tp1.

can_tp1_service_register

Syntax

```
can_tp1_canService* can_tp1_service_register(
    const can_tp1_canChannel* canCh,
    const UInt32 service_type);
```

Include file

CanTp1.h

Purpose

To register the service function.

Description

Use `can_tp1_service_read` to read a registered service specified by the `service_type` parameter.

Parameters

canCh Specifies the pointer to the `can_tp1_canChannel` structure.
service_type Specifies the service to be installed. For additional information, see the `type` parameter of `can_tp1_canService` structure. You can use the bitwise OR operator to combine several services.

Return value

canService This function returns the pointer to the `can_tp1_canService` structure.

Messages

The following messages are defined:

ID	Type	Message	Description
101	Error	can_tp1_service_register(x,..) memory allocation error on master	Memory allocation error. No free memory on the master.
102	Error	can_tp1_service_register(x,..) queue: Illegal communication queue.	There is no communication channel with this queue number.
103	Error	can_tp1_service_register(x,..) index: illegal function index	The index does not exist in the command table and is not equal to CAN_TP1_AUTO_INDEX.
104	Error	can_tp1_service_register(x,..) queue: master to slave overflow	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted.
106	Error	can_tp1_service_register(x,..) slave: not responding	The slave did not finish the initialization of the communication within one second.
107	Error	can_tp1_service_register(x,..) slave: memory allocation error	Memory allocation error on the slave. There are too many functions registered.
108	Error	can_tp1_service_register(x,..) queue: slave to master overflow	Not enough memory space between the slave write pointer and the master read pointer. The slave tries to write data to a filled queue. To prevent this error deactivate all messages with <code>can_tp1_msg_sleep</code> or <code>can_tp1_channel_all_sleep</code> when registering messages or services.
152	Error	can_tp1_service_register(x,..) canCh: the CAN channel wasn't initialized	This message is displayed if: <ul style="list-style-type: none"> ▪ You try to register a CAN message on an uninitialized CAN channel. You try to register a CAN service on an uninitialized CAN channel. ▪ You try to start an uninitialized CAN channel with <code>can_tp1_channel_start</code>. Use <code>can_tp1_channel_init</code> to initialize the CAN channel.

Example

For a detailed example of how to use this function, refer to [Example of Using Service Functions](#) on page 885.

```
can_tp1_canService* service;
...
service = can_tp1_can_service_register(txCh,
                                         CAN_TP1_SERVICE_TX_OK |
                                         CAN_TP1_SERVICE_TXQUEUE_OVERFLOW_COUNT );
```

Related topics**References**

can_tp1_canService.....	809
can_tp1_channel_all_sleep.....	822
can_tp1_channel_init.....	818
can_tp1_msg_sleep.....	860
can_tp1_service_read.....	873
can_tp1_service_request.....	872

can_tp1_service_request

Syntax

```
Int32 can_tp1_service_request(
    const can_tp1_canService* service);
```

Include file

CanTp1.h

Purpose

To request the service information from the slave can_tp1. Use `can_tp1_service_read` to read the registered service.

Description

Use the returned handle from the function `can_tp1_service_register` on page 870 to call this function.

Parameters

service Specifies the pointer to the `can_tp1_canService` structure.

Return value

This function returns the error code. The following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_BUFFER_OVERFLOW	Not enough memory space between the master write pointer and the slave read pointer. The operation is aborted. Repeat the function until it returns CAN_TP1_NO_ERROR.
CAN_TP1_NO_DATA	<code>can_tp1_service_request</code> was called without registering the function with <code>can_tp1_service_register</code> or an empty service structure was handled.

Example

For an example of how to use this function, refer to [Example of Using Service Functions](#) on page 885.

Related topics

References

can_tp1_canService.....	809
can_tp1_service_read.....	873
can_tp1_service_register.....	870

can_tp1_service_read

Syntax

```
Int32 can_tp1_service_read(
    can_tp1_canService* service);
```

Include file

CanTp1.h

Purpose

To read the service information from the slave can_tp1.

Description

Prior to this, this information must have been requested by the master calling the `can_tp1_service_request` function that asks for the service information from the slave can_tp1.

Use the returned handle from the `can_tp1_service_register` function.

Parameters

service Specifies the pointer to the updated `can_tp1_canService` structure. The following data will be updated if available: busstatus, stdmask, extmask, msg_mask15, tx_ok, rx_ok, crc_err, ack_err, form_err, stuffbit_err, bit1_err, bit0_err, rx_lost, data_lost, version, mailbox_err, txqueue_overflowcnt_std, txqueue_overflowcnt_ext.

Return value

This function returns the error code. The following symbols are predefined:

Symbols	Meaning
CAN_TP1_NO_ERROR	The function was performed without error.
CAN_TP1_NO_DATA	No data was updated.
CAN_TP1_DATA_LOST	The input data of a previous request for the specified function was overwritten.

Example

For an example of how to use this function, refer to [Example of Using Service Functions](#) on page 885.

```
can_tp1_canService* service;
...
service = can_tp1_service_register(txCh,
    CAN_TP1_SERVICE_TX_OK |
    CAN_TP1_SERVICE_TXQUEUE_OVERFLOW_COUNT);
can_tp1_service_request( service );
can_tp1_service_read( service );
/* output */
txok = service->tx_ok;
queueoverflow = service->txqueue_overflowcnt_std;
```

Related topics

References

can_tp1_canService.....	809
can_tp1_service_register.....	870
can_tp1_service_request.....	872

CAN Subinterrupt Handling

Where to go from here

Information in this section

[Defining a Callback Function](#).....875

The callback function is a function that performs the action(s) that you define for a given subinterrupt.

[can_tp1_subint_handler_install](#).....876

To install a subinterrupt handler for all CAN interrupts.

Defining a Callback Function

Callback function

The callback function is a function that performs the action(s) that you define for a given subinterrupt. The callback function must be installed with the `can_tp1_subint_handler_install` function.

Each time a CAN subinterrupt occurs, the subinterrupt handling then passes the information to the callback function.

Defining a callback function

Define your callback function as follows:

```
void can_callback_fcn(void* subint_data, Int32 subint);
```

with the parameters

subint_data Pointer to the board index of the related board within the range 0 ... 15

subint Subinterrupt number within the range 0 ... 14

Note

The last subinterrupt number to be generated is always "-1". This value indicates that there are no more pending subinterrupts.

Related topics

References

[can_tp1_subint_handler_install](#).....876

can_tp1_subint_handler_install

Syntax

```
can_tp1_subint_handler_t can_tp1_subint_handler_install(  
    const UInt32 ModuleAddr,  
    const can_tp1_subint_handler_t handler);
```

Include file

CanTp1.h

Purpose

To install a subinterrupt handler for all CAN interrupts.

Parameters

ModuleAddr Specifies the address of the CAN module.

You can use the following predefined symbol:

Predefined Symbol	CAN Type1 Module Number (used by RTI)
CAN_TP1_1_MODULE_ADDR	1

handler Specifies the pointer to your callback function.

For information on defining a callback function, refer to [Defining a Callback Function](#) on page 875.

Return value

This function returns the following value:

Symbol	Meaning
can_tp1_subint_handler_t	Pointer to the previously installed callback function

Example

For an example of how to use this function, refer to [Example of Using Subinterrupts](#) on page 883.

Related topics

Basics

[Defining a Callback Function](#)..... 875

Utilities

Introduction Information on setting the time base to a defined value, clearing CAN data on the master, and reading the current error code.

Where to go from here	Information in this section
	<p>can_tp1_all_data_clear.....877 To clear the data buffer of the master.</p>
	<p>can_tp1_error_read.....878 To read the current error of the slave can_tp1 from the dual-port memory.</p>

can_tp1_all_data_clear

Syntax	<code>void can_tp1_all_data_clear(const UInt32 ModuleAddr);</code>				
Include file	CanTp1.h				
Purpose	To clear the data buffer of the master. This is required by the RTI environment to clear all data when restarting the simulation.				
Parameters	<p>ModuleAddr Specifies the address of the CAN module. You can use the following predefined symbol:</p> <table border="1"> <thead> <tr> <th>Predefined Symbol</th> <th>CAN Type1 Module Number (used by RTI)</th> </tr> </thead> <tbody> <tr> <td>CAN_TP1_1_MODULE_ADDR</td> <td>1</td> </tr> </tbody> </table>	Predefined Symbol	CAN Type1 Module Number (used by RTI)	CAN_TP1_1_MODULE_ADDR	1
Predefined Symbol	CAN Type1 Module Number (used by RTI)				
CAN_TP1_1_MODULE_ADDR	1				
Return value	None				
Example	<code>can_tp1_all_data_clear(CAN_TP1_1_MODULE_ADDR)</code>				

Related topics**References**

[can_tp1_msg_clear.....](#) 865

can_tp1_error_read

Syntax

```
Int32 can_tp1_error_read(
    const UInt32 ModuleAddr,
    const Int32 queue);
```

Include file

CanTp1.h

Purpose

To read the current error of the slave can_tp1 from the dual-port memory.

Parameters**ModuleAddr** Specifies the address of the CAN module.

You can use the following predefined symbol:

Predefined Symbol	CAN Type1 Module Number (used by RTI)
CAN_TP1_1_MODULE_ADDR	1

queue Specifies the communication channel within the range 0 ... 6.**Return value**

This function returns the error code; the following symbols are predefined:

Predefined Symbol	Meaning
CAN_TP1_NO_ERROR	No error on the slave can_tp1.
CAN_TP1_SLAVE_ALLOC_ERROR	Memory allocation error on the slave can_tp1. There are too many functions registered.
CAN_TP1_SLAVE_BUFFER_OVERFLOW	Not enough memory space between the slave write pointer and the master read pointer.
CAN_TP1_INIT_ACK	Acknowledge code. This is no error.
CAN_TP1_SLAVE_UNDEF_ERROR	Undefined error. An error that cannot be assigned to one of the previous errors.

Example

```
#define QUEUE0 0
Int32 slave_error;
slave_error = can_tp1_error_read(CAN_TP1_1_MODULE_ADDR, QUEUE0);
/*
/* error handling */
/* */
```

Examples of Using CAN

Introduction	Examples of how to use the CAN functions.
---------------------	---

Where to go from here	Information in this section
	Example of Handling Transmit and Receive Messages880 Shows how to register a transmit and a receive message.
	Example of Handling Request and Remote Messages881 Shows how to register a request and a remote message.
	Example of Using Subinterrupts883 Shows how to register messages which can generate a subinterrupt.
	Example of Using Service Functions885 Shows how to use the service functions <code>CAN_TP1_SERVICE_TX_OK</code> and <code>CAN_TP1_SERVICE_RX_OK</code> .
	Example of Receiving Different Message IDs886 Shows how to set up a CAN controller to receive the message IDs 0x100 ... 0xFF via one message queue.

Example of Handling Transmit and Receive Messages

Example	This example shows how to register a transmit and a receive message.
----------------	--

After a delay of 4.0 seconds, the transmit message is sent periodically every 1.0 seconds. If you connect the two CAN channels with each other, you can receive the transmitted CAN message on the other CAN channel. After the CAN message is received successfully, an info message is sent to the message module.

```

1 #include <Brtenv.h>
2 can_tp1_canChannel* txCh;
3 can_tp1_canChannel* rxCh;
4 can_tp1_canMsg* txMsg;
5 can_tp1_canMsg* rxMsg;
6 UInt32 txMsgData[8] = {1,2,3,4,5,6,7,8};
7 main()
8 {
9     RTLlib_INIT();
10    can_tp1_communication_init(CAN_TP1_1_MODULE_ADDR,
11        CAN_TP1_INT_DISABLE);
12    txCh = can_tp1_channel_init(CAN_TP1_1_MODULE_ADDR, 0,
13        500000,
14        CAN_TP1_STD,
15        CAN_TP1_NO_SUBINT);
16    rxCh = can_tp1_channel_init(CAN_TP1_1_MODULE_ADDR, 1,
```

```

17         500000,
18         CAN_TP1_STD,
19         CAN_TP1_NO_SUBINT);
20 txMsg = can_tp1_msg_tx_register(txCh,
21         2,
22         0x123,
23         CAN_TP1_STD,
24         CAN_TP1_TIMECOUNT_INFO,
25         CAN_TP1_NO_SUBINT,
26         4.0,
27         1.0,
28         CAN_TP1_TIMEOUT_NORMAL);
29 rxMsg = can_tp1_msg_rx_register(rxCh,
30         3,
31         0x123,
32         CAN_TP1_STD,
33         CAN_TP1_DATA_INFO | CAN_TP1_TIMECOUNT_INFO,
34         CAN_TP1_NO_SUBINT);
35 can_tp1_msg_write(txMsg, 8, txMsgData);
36 can_tp1_channel_start(rxCh, CAN_TP1_INT_DISABLE);
37 can_tp1_channel_start(txCh, CAN_TP1_INT_DISABLE);
38 for(;;)
39 {
40     can_tp1_msg_read(txMsg);
41     if (txMsg->processed == CAN_TP1_PROCESSED)
42     {
43         msg_info_printf(MSG_SM_RTLIB, 0,
44             "TX CAN message, time: %f, deltatime: %f ",
45             txMsg->timestamp, txMsg->deltatime);
46     }
47     can_tp1_msg_read(rxMsg);
48     if (rxMsg->processed == CAN_TP1_PROCESSED)
49     {
50         msg_info_printf(MSG_SM_RTLIB, 0,
51             "RX CAN message, time: %f, deltatime: %f ",
52             rxMsg->timestamp, rxMsg->deltatime);
53     }
54     RTLIB_BACKGROUND_SERVICE();
55 }
56 }
```

Related topics**Examples**

Example of Handling Request and Remote Messages.....	881
Example of Receiving Different Message IDs.....	886
Example of Using Service Functions.....	885
Example of Using Subinterrupts.....	883

Example of Handling Request and Remote Messages

Example

This example shows how to register a request and a remote message.

After a delay of 4.0 seconds, the request message is sent periodically every 2.0 seconds. If you connect the two CAN channels with each other you can receive the request message on the other CAN channel. After the requested data is received successfully, an info message is sent to the message module.

```

1 #include <Brtenv.h>
2
3 can_tp1_canChannel* rqCh;
4 can_tp1_canChannel* rmCh;
5 can_tp1_canMsg* rqtxMsg;
6 can_tp1_canMsg* rqrxFMsg;
7 can_tp1_canMsg* rmMsg;
8 UInt32 rmMsgData[8] = {1,2,3,4,5,6,7,8};
9
10 main()
11 {
12     RTLlib_INIT();
13
14     can_tp1_communication_init(CAN_TP1_1_MODULE_ADDR,
15                                 CAN_TP1_INT_DISABLE);
16
17     rqCh = can_tp1_channel_init(CAN_TP1_1_MODULE_ADDR, 0,
18                                 500000,
19                                 CAN_TP1_STD,
20                                 CAN_TP1_NO_SUBINT);
21
22     rmCh = can_tp1_channel_init(CAN_TP1_1_MODULE_ADDR, 1,
23                                 500000,
24                                 CAN_TP1_STD,
25                                 CAN_TP1_NO_SUBINT);
26
27     rqtxMsg = can_tp1_msg_rqtx_register(rqCh,
28                                         2,
29                                         0x123,
30                                         CAN_TP1_STD,
31                                         CAN_TP1_TIMECOUNT_INFO,
32                                         CAN_TP1_NO_SUBINT,
33                                         4.0,
34                                         2.0,
35                                         CAN_TP1_TIMEOUT_NORMAL);
36
37     rqrxFMsg = can_tp1_msg_rqrxFMsg_register(rqtxMsg,
38                                              CAN_TP1_DATA_INFO | CAN_TP1_TIMECOUNT_INFO,
39                                              CAN_TP1_NO_SUBINT);
40
41     rmMsg = can_tp1_msg_rm_register(rmCh,
42                                     3,
43                                     0x123,
44                                     CAN_TP1_STD,
45                                     CAN_TP1_TIMECOUNT_INFO,
46                                     CAN_TP1_NO_SUBINT);
47
48     can_tp1_msg_write(rmMsg, 8, rmMsgData);
49
50     can_tp1_msg_rqtx_activate(rqtxMsg);
51
52     can_tp1_channel_start(rqCh, CAN_TP1_INT_DISABLE);
53
54     can_tp1_channel_start(rmCh, CAN_TP1_INT_DISABLE);
55

```

```

56    for(;;)
57    {
58        can_tp1_msg_read(rqrxF);
59        can_tp1_msg_read(rqtxF);
60        can_tp1_msg_read(rmF);
61
62        if (rqrxF->processed == CAN_TP1_PROCESSED)
63        {
64            msg_info_printf(MSG_SM_RTLIB, 0,
65                            "QRX CAN message, time: %f,deltatime: %f ",
66                            rqrxF->tstamp, rqrxF->deltatime);
67        }
68
69        if (rqtxF->processed == CAN_TP1_PROCESSED)
70        {
71            msg_info_printf(MSG_SM_RTLIB, 0,
72                            "RQTX CAN message, time: %f,deltatime: %f ",
73                            rqtxF->tstamp, rqtxF->deltatime);
74        }
75
76        if (rmF->processed == CAN_TP1_PROCESSED)
77        {
78            msg_info_printf(MSG_SM_RTLIB, 0,
79                            "RM CAN message, time: %f,deltatime: %f ",
80                            rmF->tstamp, rmF->deltatime);
81        }
82
83        RLIB_BACKGROUND_SERVICE();
84
85    }
86 }
```

Related topics**Examples**

Example of Handling Transmit and Receive Messages.....	880
Example of Receiving Different Message IDs.....	886
Example of Using Service Functions.....	885
Example of Using Subinterrupts.....	883

Example of Using Subinterrupts

Example

This example shows how to register messages that can generate a subinterrupt.

The CAN controller is started and a CAN message is sent immediately. If the CAN message was sent successfully, a subinterrupt is generated to call the installed callback function.

The callback function in this example evaluates the specified subinterrupt and sends the CAN message again with a time delay of 0.1 s.

After the CAN message is received, another subinterrupt is generated to read the CAN message and pass an info message to the message module.

Note

The CAN channels 0 and 1 have to be connected.

```

1 #include <Brtenv.h>
2 #define tx_subint 2
3 #define rx_subint 3
4 can_tp1_canChannel* txCh;
5 can_tp1_canChannel* rxCh;
6 can_tp1_canMsg* txMsg;
7 can_tp1_canMsg* rxMsg;
8 UInt32 txMsgData[8] = { 1,2,3,4,5,6,7,8 };
9 void can_user_callback(const UInt32 ModuleAddr, const Int32 subint)
{
10     switch(subint)
11     {
12         case tx_subint:
13             txMsgData[0] = (txMsgData[0]+1) & 0xFF;
14             /* send the message delayed */
15             can_tp1_msg_send( txMsg, 8, txMsgData, 0.1);
16             msg_info_printf(MSG_SM_RTLIB, 0, "TX Subint:%d", subint);
17             break;
18         case rx_subint:
19             /* read the message from the communication buffer */
20             can_tp1_msg_read(rxMsg);
21             msg_info_printf(MSG_SM_RTLIB,
22                             0,
23                             "RX Subint:%d, time: %fs, deltatime: %fs data[0]: %x",
24                             subint,
25                             rxMsg->timestamp,
26                             rxMsg->deltatime,
27                             rxMsg->data[0]);
28             break;
29         default:
30             break;
31     }
32 }
33 main()
34 {
35     RTLlib_INIT();
36     can_tp1_communication_init(CAN_TP1_1_MODULE_ADDR,
37                                 CAN_TP1_INT_ENABLE);
38     can_tp1_subint_handler_install(CAN_TP1_1_MODULE_ADDR,
39                                    can_user_callback);
40     txCh = can_tp1_channel_init (CAN_TP1_1_MODULE_ADDR, 1, 500000,
41                                 CAN_TP1_STD,
42                                 CAN_TP1_NO_SUBINT);
43     txMsg = can_tp1_msg_tx_register(txCh,
44                                     0,
45                                     0x123,
46                                     CAN_TP1_STD,
47                                     CAN_TP1_NO_INFO,
48                                     tx_subint,
49                                     0.0,
50                                     0.0,
51                                     CAN_TP1_TIMEOUT_NORMAL);
52     rxCh = can_tp1_channel_init(CAN_TP1_1_MODULE_ADDR,
53

```

```

54          0,
55          500000,
56          CAN_TP1_STD,
57          CAN_TP1_NO_SUBINT);
58 rxMsg = can_tp1_msg_rx_register(rxCh,
59          0,
60          0x123,
61          CAN_TP1_STD,
62          CAN_TP1_DATA_INFO |
63          CAN_TP1_TIMECOUNT_INFO,
64          rx_subint);
65 can_tp1_channel_start(rxCh, CAN_TP1_INT_DISABLE);
66 can_tp1_channel_start(txCh, CAN_TP1_INT_DISABLE);
67 can_tp1_msg_send( txMsg, 8, txMsgData, 0.0);
68 RTLlib_INT_ENABLE();
69 for(;;)
70 {
71     RTLlib_BACKGROUND_SERVICE();
72 }
73 }
```

Related topics**Examples**

Example of Handling Request and Remote Messages.....	881
Example of Handling Transmit and Receive Messages.....	880
Example of Receiving Different Message IDs.....	886
Example of Using Service Functions.....	885

Example of Using Service Functions

Example

This example shows how to use the service functions CAN_TP1_SERVICE_TX_OK and CAN_TP1_SERVICE_RX_OK.

Note

No message is installed on the can_tp1 in this example.

```

1 #include <Brtenv.h>
2 can_tp1_canChannel* canCh0;
3 can_tp1_canChannel* canCh1;
4 can_tp1_canService* txokServ;
5 can_tp1_canService* rxokServ;
6 main()
7 {
8     RTLlib_INIT();
9     can_tp1_communication_init(CAN_TP1_1_MODULE_ADDR,
10                             CAN_TP1_INT_DISABLE);
11    canCh0 = can_tp1_channel_init(CAN_TP1_1_MODULE_ADDR, 0,
12                                500000, CAN_TP1_STD, CAN_TP1_NO_SUBINT);
13    canCh1 = can_tp1_channel_init(CAN_TP1_1_MODULE_ADDR, 1,
14                                500000, CAN_TP1_STD, CAN_TP1_NO_SUBINT);
```

```

15  /* register the tx_ok function which delivers the count */
16  /* of the can_tp1 tx-ok counter for CAN channel 0 */
17  txokServ = can_tp1_service_register(canCh0,
18          CAN_TP1_SERVICE_TX_OK);
19  /* register the rx_ok function which delivers the count */
20  /* of the can_tp1 rx-ok counter for CAN channel 1 */
21  rxokServ = can_tp1_service_register(canCh1,
22          CAN_TP1_SERVICE_RX_OK);
23  for(;;)
24  {
25      /* request the tx-ok counter from the slave can_tp1 */
26      can_tp1_service_request(txokServ);
27      /* request the rx-ok counter from the slave can_tp1 */
28      can_tp1_service_request(rxokServ);
29      /* read the tx-ok counter from the slave can_tp1 */
30      /* the data will be available in txokServ->data0 */
31      can_tp1_service_read(txokServ);
32      /* read the rx-ok counter from the slave */
33      /* the data will be available in rxokServ->data0 */
34      can_tp1_service_read(rxokServ);
35      RTLIB_BACKGROUND_SERVICE();
36  }
37 }
```

Related topics**Examples**

Example of Handling Request and Remote Messages.....	881
Example of Handling Transmit and Receive Messages.....	880
Example of Receiving Different Message IDs.....	886
Example of Using Subinterrupts.....	883

Example of Receiving Different Message IDs

Example

This example shows you how to set up a CAN controller to receive the message IDs 0x100 ... 0xFF via one message queue.

```

1 #include <Brtenv.h>
2 can_tp1_canChannel* rxCh;
3 can_tp1_canMsg* canMonitor;
4 UInt32 data[8];
5 UInt32 mask = 0xFFFFF00;
6 UInt32 queue_size = 64;
7 main()
8 {
9     RTLIB_INIT();           /* initialize hardware system */
10    can_tp1_communication_init(CAN_TP1_1_MODULE_ADDR,
11                                CAN_TP1_INT_DISABLE);
12    rxCh = can_tp1_channel_init(CAN_TP1_1_MODULE_ADDR,
13                                0,
14                                500000,
15                                CAN_TP1_STD,
16                                CAN_TP1_NO_SUBINT);
```

```

17 canMonitor = can_tp1_msg_rx_register (rxCh,
18     1,
19     0x100,
20     CAN_TP1_STD,
21     CAN_TP1_TIMECOUNT_INFO |
22     CAN_TP1_MSG_INFO,
23     CAN_TP1_NO_SUBINT);
24 can_tp1_msg_set(canMonitor,
25     CAN_TP1_MSG_MASK,
26     &mask);
27 can_tp1_msg_set(canMonitor,
28     CAN_TP1_MSG_QUEUE,
29     &queue_size);
30 can_tp1_channel_start(rxCh, CAN_TP1_INT_DISABLE);
31 for(;;)
32 {
33     can_tp1_msg_read(canMonitor);
34     if (canMonitor->processed == CAN_TP1_PROCESSED)
35     {
36         msg_info_printf(0,0,"id: %d time: %f",
37                         canMonitor->identifier, canMonitor->timestamp);
38     }
39     RTLIB_BACKGROUND_SERVICE();
40 }
41 }
```

Related topics**Examples**

Example of Handling Request and Remote Messages.....	881
Example of Handling Transmit and Receive Messages.....	880
Example of Using Service Functions.....	885
Example of Using Subinterrupts.....	883

FPGA Module Access Functions

Purpose

The board's RTLib provides functions to access the I/O FPGA Type 1 module for initialization, read and write accesses, and interrupt handling.

There is no function required for programming the FPGA module because of the loading mechanism of MicroLabBox.

Where to go from here

Information in this section

FPGA Initialization	890
---	-----

Before you can use the I/O FPGA application, you have to perform the initialization process.

Identification Functions	892
--	-----

Functions to get the identifier of the various components concerning the I/O FPGA application.

Interrupt Functions	896
---	-----

Functions to access the interrupt sources of the I/O FPGA application.

Data Exchange Functions	901
---	-----

Functions to exchange data between the FPGA application running on the I/O FPGA application and the processor board.

Information in other sections

[FPGA Support \(MicroLabBox Features](#)

MicroLabBox's I/O FPGA module is used for the standard I/O features or for custom FPGA applications.

FPGA Initialization

Introduction

Before you can use the I/O FPGA application, you have to perform the initialization process.

IoFpga_init

Syntax

```
void IoFpga_init(
    ULONG64* fpga_appl_id)
```

Include file

`IoFpga.h`

Purpose

To initialize the I/O FPGA application with default settings.

Description

This function initializes the I/O FPGA application:

- Checks whether the processor application is compatible with the FPGA application.

Parameters

fpga_appl_id Specifies the pointer to the variable containing the identifier of the FPGA application that is generated by the FPGA build process.

Return value

None

Messages

The following messages are defined:

ID	Type	Message	Description
201	Error	IoFpga_init(): Invalid module address 0x??	The value of the module address parameter is not a valid module address. This error may be caused if the intermodule bus connection of the I/O board is missing. Check the connection.
300	Error	IoFpga_init(0x??): Board not found!	No IOFPGA module could be found at the specified module address.
301	Error	IoFpga_init(0x??): Board is not responding! Hardware reset failed.	The IOFPGA module has not booted after power-up.
302	Error	IoFpga_init(0x??): No FPGA application loaded.	The module's FPGA is not configured with an application.

ID	Type	Message	Description
303	Error	IoFpga_init(0x??): FPGA is not responding! FPGA application initialization failed.	The module's FPGA was configured but failed to boot and initialize the FPGA application.
304	Error	IoFpga_init(0x??): Piggyback family 0x??? is not matching FPGA_TP1 FPGA framework family 0x????!	The piggyback family does not match the IOFPGA FPGA framework family.
305	Error	IoFpga_init(0x??): Piggyback revision 0x??? is not matching FPGA_TP1 FPGA framework revision 0x????	The piggyback revision does not match the IOFPGA FPGA framework revision.
306	Error	IoFpga_init(0x??): FPGA application ID 0x??? expected by processor application does not match ID 0x??? of running FPGA application!	The process and FPGA applications are incompatible due to non matching FPGA application IDs.
307	Error	IoFpga_init(0x??): Fatal hardware error of IOFPGA module!	The IOFPGA status indicates a fatal hardware error.

Example

This example shows how to initialize the I/O FPGA application:

```
void main(void)
{
    ULONG64 fpga_appl_id = {0x48C01235, 0x592DC2};

    init();
    IoFpga_init(&fpga_appl_id);
    ...
}
```

Identification Functions

Introduction	Functions to get the identifiers of the various components associated with the I/O FPGA application.
---------------------	--

Where to go from here	Information in this section
	IoFpga_get_board_rev892 To read the I/O FPGA application revision.
	IoFpga_get_fw_id893 To read the FPGA framework identifier.
	IoFpga_get_appl_id894 To read the FPGA application identifier.
	IoFpga_get_appl_id_string895 To read the description of the running FPGA application.

IoFpga_get_board_rev

Syntax	<pre>void IoFpga_get_board_rev(UInt32* major, UInt32* minor)</pre>
Include file	<code>IoFpga.h</code>
Purpose	To read the I/O FPGA application revision.
Description	This function reads out the revision of the I/O FPGA application. It consists of a major and a minor revision number.
Parameters	<p>major Specifies the pointer to the variable containing the major revision number of the I/O FPGA application.</p> <p>minor Specifies the pointer to the variable containing the minor revision number of the I/O FPGA application.</p>

Return value	None
Example	This example shows how to get the major and minor board revision numbers: <pre>IoFpga_get_board_rev(&major, &minor);</pre>
Related topics	References

IoFpga_get_appl_id.....	894
IoFpga_get_appl_id_string.....	895
IoFpga_get_fw_id.....	893
IoFpga_init.....	890

IoFpga_get_fw_id

Syntax	<pre>void IoFpga_get_fw_id(UInt32* family, UInt32* major, UInt32* minor)</pre>
Include file	<code>IoFpga.h</code>
Purpose	To read the FPGA framework identifier.
Description	This function reads the identifier of the FPGA framework. It consists of a family identification number, the major revision number and the minor revision number. The framework corresponds to the firmware of the FPGA. For information on the framework, refer to FPGA Support (MicroLabBox Features) .
Parameters	<p>family Specifies the pointer to the variable containing the family identification number of the FPGA framework.</p> <p>major Specifies the pointer to the variable containing the major revision number of the FPGA framework.</p> <p>minor Specifies the pointer to the variable containing the minor revision number of the FPGA framework.</p>
Return value	None

Example

This example shows how to get the framework identifier:

```
IoFpga_get_fw_id(&family, &major, &minor);
```

Related topics**References**

IoFpga_get_appl_id.....	894
IoFpga_get_appl_id_string.....	895
IoFpga_get_board_rev.....	892
IoFpga_init.....	890

IoFpga_get_appl_id

Syntax

```
void IoFpga_get_appl_id(  
    ULong64* appl_id)
```

Include file

`IoFpga.h`

Purpose

To read the FPGA application identifier.

Description

This function reads the identifier of the running FPGA application.

Parameters

appl_id Specifies the pointer to the variable containing the identifier of the FPGA application.

Return value

None

Example

This example shows how to get the identifier of the running application:

```
IoFpga_get_appl_id(&appl_id);
```

Related topics**References**

IoFpga_get_appl_id_string.....	895
IoFpga_get_board_rev.....	892
IoFpga_get_fw_id.....	893
IoFpga_init.....	890

IoFpga_get_appl_id_string

Syntax

```
void IoFpga_get_appl_id_string(  
    char* id_string)
```

Include file

`IoFpga.h`

Purpose

To read the description of the running FPGA application.

Description

This function reads a string that describes the running FPGA application.

Parameters

id_string Specifies the pointer to the variable containing the description of the FPGA application. It can have a size of 256 characters.

Return value

None

Example

This example shows how to get the description of the running FPGA application:

```
...  
char id_string[256];  
...  
IoFpga_get_appl_id_string(id_string);
```

Related topics**References**

IoFpga_get_appl_id.....	894
IoFpga_get_board_rev.....	892
IoFpga_get_fw_id.....	893
IoFpga_init.....	890

Interrupt Functions

Introduction	Functions to access the interrupt sources of the I/O FPGA application.
---------------------	--

Where to go from here	Information in this section
	IoFpga_enable_int896 To enable the interrupt channels of the I/O FPGA application.
	IoFpga_disable_int897 To disable the interrupt channels of the I/O FPGA application.
	IoFpga_ack_int898 To acknowledge the pending interrupts of the I/O FPGA application.
	IoFpga_register_isr899 To register an interrupt service routine.

IoFpga_enable_int

Syntax	<code>void IoFpga_enable_int(UInt32 int_src)</code>
Include file	<code>IoFpga.h</code>
Purpose	To enable the interrupt channels of the I/O FPGA application.
Description	The I/O FPGA application provides 32 interrupt channels that you must enable before you can use them as interrupt sources. Each channel can be enabled separately.
Parameters	int_src Specifies the bit mask of interrupt channels to be enabled. If you want to enable more than one interrupt channel, you can combine the following predefined symbols by using the logical operator OR.

Symbols	Meaning
IOFPGA_INT_SRC_1	Interrupt on channel 1
...	...
IOFPGA_INT_SRC_32	Interrupt on channel 32

Return value None

Example This example shows how to enable the interrupts on channels 1 and 3:

```
IoFpga_enable_int(IOFPGA_INT_SRC_1|IOFPGA_INT_SRC_3);
```

Related topics References

IoFpga_ack_int.....	898
IoFpga_disable_int.....	897
IoFpga_init.....	890

IoFpga_disable_int

Syntax

```
void IoFpga_disable_int(  
    UInt32 int_src)
```

Include file IoFpga.h

Purpose To disable the interrupt channels of the I/O FPGA application.

Description The I/O FPGA application provides 32 interrupt channels. If you have enabled them by using the **IoFpga_enable_int** function, you can disable them again.

Parameters **int_src** Specifies the bit mask of interrupt channels to be disabled. If you want to disable more than one interrupt channel, you can combine the following predefined symbols by using the logical operator OR.

Symbols	Meaning
IOFPGA_INT_SRC_1	Interrupt on channel 1
...	...
IOFPGA_INT_SRC_32	Interrupt on channel 32

Return value None

Example This example shows how to disable the interrupts of channels 1 and 3:

```
IoFpga_disable_int(IOFPGA_INT_SRC_1|IOFPGA_INT_SRC_3);
```

Related topics References

IoFpga_ack_int.....	898
IoFpga_enable_int.....	896
IoFpga_init.....	890

IoFpga_ack_int

Syntax

```
__INLINE void IoFpga_ack_int(  
    UInt32 int_src)
```

Include file IoFpga.h

Purpose To acknowledge the pending interrupts of the I/O FPGA application.

Description This function acknowledges the interrupt on the specified channel(s). The acknowledgement resets the associated flag in the interrupt flag register. For example, this avoids interrupts directly after enabling them.

Parameters **int_src** Specifies the bit mask of the interrupt channels to be acknowledged. If you want to acknowledge more than one interrupt channel, you can combine the following predefined symbols by using the logical operator OR.

Symbols	Meaning
IOFPGA_INT_SRC_1	Interrupt on channel 1
...	...
IOFPGA_INT_SRC_32	Interrupt on channel 32

Return value None

Example This example shows how to acknowledge the interrupts on channels 1 and 3:

```
IoFpga_ack_int(IOFPGA_INT_SRC_1|IOFPGA_INT_SRC_3);
```

Related topics References

IoFpga_disable_int.....	897
IoFpga_enable_int.....	896
IoFpga_init.....	890

IoFpga_register_isr

Syntax

```
__INLINE void IoFpga_register_isr(
    IoFpga_TIntHandler pIsr,
    UInt32 int_no)
```

Include file IoFpga.h

Purpose To register an interrupt service routine.

Description An interrupt service routine (ISR) is executed when a certain interrupt occurs. With this function, you can connect the interrupt service routine to the specified interrupt.

Parameters **pIsr** Specifies the pointer to the interrupt service routine (ISR) to be registered.

int_no Specifies the interrupt the interrupt service routine is to be registered for.

Symbols	Meaning
IOFPGA_INT_SRC_1	Interrupt on channel 1
...	...
IOFPGA_INT_SRC_32	Interrupt on channel 32

Return value None

Example This example shows how to register the function `isr_0` for interrupt 1:

```
RTLIB_INT_ENABLE();
IoFpga_register_isr(isr_0, IOFPGA_INT_SRC_1);
IoFpga\_enable\_int on page 896(IOFPGA_INT_SRC_1);
```

Related topics

References

<code>IoFpga_ack_int</code>	898
<code>IoFpga_disable_int</code>	897
<code>IoFpga_enable_int</code>	896
<code>IoFpga_init</code>	890

Data Exchange Functions

Introduction Functions to exchange data between the FPGA application running on the I/O FPGA application and the processor board.

Where to go from here	Information in this section
	<p>IoFpga_read_reg..... 902 To read data from an exchange register of the I/O FPGA application.</p> <p>IoFpga_read_reg64..... 903 To read 64-bit data from an exchange register of the I/O FPGA application.</p> <p>IoFpga_write_reg..... 905 To write data to an exchange register of the I/O FPGA application.</p> <p>IoFpga_write_reg64..... 906 To write 64-bit data to an exchange register of the I/O FPGA application.</p> <p>IoFpga_read_reg_grp..... 908 To read from a group of exchange registers of the I/O FPGA application.</p> <p>IoFpga_read_reg_grp_mixed..... 910 To read from a group of 32-bit and 64-bit exchange registers of the I/O FPGA application.</p> <p>IoFpga_write_reg_grp..... 912 To write to a group of exchange registers of the I/O FPGA application.</p> <p>IoFpga_write_reg_grp_mixed..... 914 To write to a group of 32-bit and 64-bit exchange registers of the I/O FPGA application.</p> <p>IoFpga_read_buf..... 917 To read data from a buffer of the I/O FPGA application.</p> <p>IoFpga_read_buf64..... 919 To read 64-bit data from a buffer of the I/O FPGA application.</p> <p>IoFpga_write_buf..... 921 To write data to a buffer of the I/O FPGA application.</p> <p>IoFpga_write_buf64..... 923 To write 64-bit data to a buffer of the I/O FPGA application.</p>

IoFpga_read_reg

Syntax

```
__INLINE void IoFpga_read_reg(
    UInt32 reg_nr,
    Float64* data,
    Float64* scaling,
    UInt32 mode)
```

Include file

`IoFpga.h`

Purpose

To read data from an exchange register of the I/O FPGA application.

Description

This function reads data from a specified exchange register and scales the data by a specified factor.

Parameters

reg_nr Specifies the number of the data register in the range 1 ... 256.

data Specifies the pointer to the variable containing the read register value.

scaling Specifies the pointer to the variable containing the scaling factor. Use one of the predefined symbols.

Symbol	Meaning
<code>IOFPGA_READ_SCALE_BIN_PT_0</code>	Scaling factor: $1/2^0 = 1.00$
<code>IOFPGA_READ_SCALE_BIN_PT_1</code>	Scaling factor: $1/2^1 = 0.50$
<code>IOFPGA_READ_SCALE_BIN_PT_2</code>	Scaling factor: $1/2^2 = 0.25$
<code>IOFPGA_READ_SCALE_BIN_PT_3</code>	Scaling factor: $1/2^3 = 0.125$
...	...
<code>IOFPGA_READ_SCALE_BIN_PT_32</code>	Scaling factor: $1/2^{32} = 2.328e^{-10}$

mode Specifies the data processing mode. Use one of the predefined symbols.

Symbol	Meaning
<code>IOFPGA_DATA_MODE_SIGNED</code>	The data is processed as a signed data type.
<code>IOFPGA_DATA_MODE_UNSIGNED</code>	The data is processed as an unsigned data type.
<code>IOFPGA_DATA_MODE_FLOAT</code>	The data is processed as floating point with a fraction width of 24, which complies with IEEE 754 standard (single). The scaling factor is ignored in this mode.

Return value	None
---------------------	------

Example	This example shows how to read data in signed mode with a scaling factor of $1/2^{16}$ from register 1:
----------------	---

```
Float64 scaling_factor = IOFPGA_READ_SCALE_BIN_PT_16;
IoFpga_read_reg(
    1,
    &data,
    &scaling_factor,
    IOFPGA_DATA_MODE_SIGNED);
```

Related topics	References
-----------------------	-------------------

IoFpga_init.....	890
IoFpga_read_buf.....	917
IoFpga_read_reg_grp.....	908
IoFpga_write_buf.....	921
IoFpga_write_reg.....	905
IoFpga_write_reg_grp.....	912

IoFpga_read_reg64

Syntax

```
__INLINE void IoFpga_read_reg64(
    UInt32 reg_nr,
    Float64* data,
    Float64* scaling,
    UInt32 mode)
```

Include file	IoFpga.h
---------------------	----------

Purpose	To read 64-bit data from an exchange register of the I/O FPGA application.
----------------	--

Description	This function reads 64-bit data from a specified exchange register and scales the data by a specified factor.
--------------------	---

Parameters	reg_nr Specifies the number of the data register in the range 1 ... 256. data Specifies the pointer to the variable containing the read register value.
-------------------	--

scaling Specifies the pointer to the variable containing the scaling factor. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_READ_SCALE_BIN_PT_0	Scaling factor: $1/2^0=1.00$
IOFPGA_READ_SCALE_BIN_PT_1	Scaling factor: $1/2^1=0.50$
IOFPGA_READ_SCALE_BIN_PT_2	Scaling factor: $1/2^2=0.25$
IOFPGA_READ_SCALE_BIN_PT_3	Scaling factor: $1/2^3=0.125$
...	...
IOFPGA_READ_SCALE_BIN_PT_64	Scaling factor: $1/2^{64}=5.421e^{-20}$

mode Specifies the data processing mode. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFPGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 53, which complies with IEEE 754 standard (double). The scaling factor is ignored in this mode.

Return value

None

Example

This example shows how to read 64-bit data in signed mode with a scaling factor of $1/2^{16}$ from register 1:

```
Float64 scaling_factor = IOFPGA_READ_SCALE_BIN_PT_16;
IoFpga_read_reg64(
    1,
    &data,
    &scaling_factor,
    IOFPGA_DATA_MODE_SIGNED);
```

Related topics**References**

IoFpga_init.....	890
IoFpga_read_buf64.....	919
IoFpga_read_reg.....	902
IoFpga_write_reg64.....	906

IoFpga_write_reg

Syntax

```
__INLINE void IoFpga_write_reg(
    UInt32 reg_nr,
    Float64* data,
    Float64* scaling,
    UInt32 mode)
```

Include file

IoFpga.h

Purpose

To write data to an exchange register of the I/O FPGA application.

Description

This function writes data to a specified exchange register and scales the data by a specified factor.

Parameters

reg_nr Specifies the number of the register in the range 1 ... 256.

data Specifies the pointer to the variable containing the data that you want to write to the register.

scaling Specifies the pointer to the variable containing the scaling factor. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_WRITE_SCALE_BIN_PT_0	Scaling factor: $2^0=1$
IOFPGA_WRITE_SCALE_BIN_PT_1	Scaling factor: $2^1=2$
IOFPGA_WRITE_SCALE_BIN_PT_2	Scaling factor: $2^2=4$
IOFPGA_WRITE_SCALE_BIN_PT_3	Scaling factor: $2^3=8$
...	...
IOFPGA_WRITE_SCALE_BIN_PT_32	Scaling factor: $2^{32}=4,294,967,296$

mode Specifies the data processing mode. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFPGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 24, which complies with IEEE 754 standard (single). The scaling factor is ignored in this mode.

Return value	None
Example	This example shows how to write data in signed mode with a scaling factor of 2^{16} to register 1:
	<pre>Float64 scaling_factor = IOFPGA_WRITE_SCALE_BIN_PT_16; IoFpga_write_reg(1, &data, &scaling_factor, IOFPGA_DATA_MODE_SIGNED);</pre>

Related topics	References												
	<table> <tr> <td>IoFpga_init.....</td> <td>890</td> </tr> <tr> <td>IoFpga_read_buf.....</td> <td>917</td> </tr> <tr> <td>IoFpga_read_reg.....</td> <td>902</td> </tr> <tr> <td>IoFpga_read_reg_grp.....</td> <td>908</td> </tr> <tr> <td>IoFpga_write_buf.....</td> <td>921</td> </tr> <tr> <td>IoFpga_write_reg_grp.....</td> <td>912</td> </tr> </table>	IoFpga_init.....	890	IoFpga_read_buf.....	917	IoFpga_read_reg.....	902	IoFpga_read_reg_grp.....	908	IoFpga_write_buf.....	921	IoFpga_write_reg_grp.....	912
IoFpga_init.....	890												
IoFpga_read_buf.....	917												
IoFpga_read_reg.....	902												
IoFpga_read_reg_grp.....	908												
IoFpga_write_buf.....	921												
IoFpga_write_reg_grp.....	912												

IoFpga_write_reg64

Syntax	<pre>__INLINE void IoFpga_write_reg64(UInt32 reg_nr, Float64* data, Float64* scaling, UInt32 mode)</pre>
Include file	<code>IoFpga.h</code>
Purpose	To write 64-bit data to an exchange register of the I/O FPGA application.
Description	This function writes 64-bit data to a specified exchange register and scales the data by a specified factor.
Parameters	<p>reg_nr Specifies the number of the register in the range 1 ... 256.</p> <p>data Specifies the pointer to the variable containing the data that you want to write to the register.</p>

scaling Specifies the pointer to the variable containing the scaling factor. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_WRITE_SCALE_BIN_PT_0	Scaling factor: $2^0=1$
IOFPGA_WRITE_SCALE_BIN_PT_1	Scaling factor: $2^1=2$
IOFPGA_WRITE_SCALE_BIN_PT_2	Scaling factor: $2^2=4$
IOFPGA_WRITE_SCALE_BIN_PT_3	Scaling factor: $2^3=8$
...	...
IOFPGA_WRITE_SCALE_BIN_PT_64	Scaling factor: $2^{64}=18,446,744,073,709,551,616$

mode Specifies the data processing mode. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFPGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 53, which complies with IEEE 754 standard (double). The scaling factor is ignored in this mode.

Return value

None

Example

This example shows how to write data in signed mode with a scaling factor of 2^{16} to register 1:

```
Float64 scaling_factor = IOFPGA_WRITE_SCALE_BIN_PT_16;
IoFpga_write_reg64(
    1,
    &data,
    &scaling_factor,
    IOFPGA_DATA_MODE_SIGNED);
```

Related topics**References**

IoFpga_init.....	890
IoFpga_read_reg64.....	903
IoFpga_write_buf64.....	923
IoFpga_write_reg.....	905

IoFpga_read_reg_grp

Syntax

```
__INLINE void IoFpga_read_reg_grp(
    UInt32 grp_nr,
    UInt32 grp_size,
    UInt32* reg_nr,
    Float64* data,
    Float64* scaling,
    UInt32* mode)
```

Include file

IoFpga.h

Purpose

To read from a group of exchange registers of the I/O FPGA application.

Description

This function reads a group of data from several specified exchange registers and scales each data value by a specified factor. The data can be assumed to be consistent, which means that each single date of the group is read from the FPGA application at the same point in time.

Parameters

grp_nr Specifies the number of the data group in the range 1 ... 63.

grp_size Specifies the group size. The number of values is in the range 1 ... 256.

reg_nr Specifies the pointer to the variable containing the number of each group register in the range 1 ... 256. The variable itself is an array of **grp_size** size.

Note

For best performance, the registers in the group should be in ascending order without gaps.

data Specifies the pointer to the variable containing the register values as an array of **grp_size** size.

scaling Specifies the pointer to the variable containing the scaling factors as an array of **grp_size** size. Use the predefined symbols.

Symbol	Meaning
IOFPGA_READ_SCALE_BIN_PT_0	Scaling factor: $1/2^0 = 1.00$
IOFPGA_READ_SCALE_BIN_PT_1	Scaling factor: $1/2^1 = 0.50$
IOFPGA_READ_SCALE_BIN_PT_2	Scaling factor: $1/2^2 = 0.25$
IOFPGA_READ_SCALE_BIN_PT_3	Scaling factor: $1/2^3 = 0.125$

Symbol	Meaning
...	...
IOFPGA_READ_SCALE_BIN_PT_32	Scaling factor: $1/2^{32} = 2.328e^{-10}$

mode Specifies the pointer to the variable containing the data modes as an array of `grp_size` size. Use the predefined symbols.

Symbol	Meaning
IOFPGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFPGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 24, which complies with IEEE 754 standard (single). The scaling factor is ignored in this mode.

Return value	None
--------------	------

Example	This example shows how to read data in signed mode from data group 1. It contains 1 value that is scaled by $1/2^{16}$:
---------	--

```
Float64 scaling_factor = IOFPGA_READ_SCALE_BIN_PT_16;
UInt32 mode = IOFPGA_DATA_MODE_SIGNED;
Float64 data;
...
IoFpga_read_reg_grp(
    1,
    1,
    &reg_nr,
    &data,
    &scaling_factor,
    &mode);
```

Related topics	References
----------------	------------

IoFpga_init.....	890
IoFpga_read_buf.....	917
IoFpga_read_reg.....	902
IoFpga_write_buf.....	921
IoFpga_write_reg.....	905
IoFpga_write_reg_grp.....	912

IoFpga_read_reg_grp_mixed

Syntax

```
__INLINE void IoFpga_read_reg_grp_mixed(
    UInt32 grp_nr,
    UInt32 grp_size[2],
    UInt32* reg_nr[2],
    Float64* data[2],
    Float64* scaling[2],
    UInt32* mode[2])
```

Include file

IoFpga.h

Purpose

To read from a group of 32-bit and 64-bit exchange registers of the I/O FPGA application.

Description

This function reads a group of data from several specified exchange registers and scales each data value by a specified factor.

You have to use this function if you are using 64-bit registers. The **IoFpga_read_reg_grp** function can only be used for 32-bit registers.

You can use **IoFpga_read_reg_grp_mixed** also for 32-bit registers and a combination of 32-bit and 64-bit registers. The data can be assumed to be consistent, which means that each single date of the group is read from the FPGA application at the same point in time.

Parameters

grp_nr Specifies the number of the data group in the range 1 ... 63.

grp_size Specifies the number of 32-bit and 64-bit registers in the group. The number of values is in the range 1 ... 256.

Item	Meaning
grp_size[0]	Specifies the number of 32-bit registers in the group.
grp_size[1]	Specifies the number of 64-bit registers in the group.

reg_nr Specifies the pointers to the variables containing the number of each group register in the range 1 ... 256. The variable itself is an array of **grp_size** size.

Note

For best performance, the registers in the group should be in ascending order without gaps.

Item	Meaning
reg_nr[0]	Specifies the register numbers of the 32-bit registers in the group.

Item	Meaning
reg_nr[1]	Specifies the register numbers of the 64-bit register in the group.

data Specifies the pointers to the variables containing the register values as an array of `grp_size` size.

Item	Meaning
data[0]	Specifies the register data of the 32-bit registers in the group.
data[1]	Specifies the register data of the 64-bit registers in the group.

scaling Specifies the pointers to the variables containing the scaling factors as an array of `grp_size` size. Use the predefined symbols.

Symbol	Meaning
IOFPGA_READ_SCALE_BIN_PT_0	Scaling factor: $1/2^0=1.00$
IOFPGA_READ_SCALE_BIN_PT_1	Scaling factor: $1/2^1=0.50$
IOFPGA_READ_SCALE_BIN_PT_2	Scaling factor: $1/2^2=0.25$
IOFPGA_READ_SCALE_BIN_PT_3	Scaling factor: $1/2^3=0.125$
...	...
IOFPGA_READ_SCALE_BIN_PT_64	Scaling factor: $1/2^{64}=5.421e^{-20}$

Item	Meaning
scaling[0]	Specifies the scaling of each 32-bit register in the group.
scaling[1]	Specifies the scaling of each 64-bit register in the group.

mode Specifies the pointers to the variables containing the data modes as an array of `grp_size` size. Use the predefined symbols.

Symbol	Meaning
IOFPGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFPGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 24 or 53, which complies with IEEE 754 standard (single or double). The scaling factor is ignored in this mode.

Item	Meaning
mode[0]	Specifies the mode of each 32-bit register in the group.
mode[1]	Specifies the mode of each 64-bit register in the group.

Return value

None

Example

This example shows how to read data in SIGNED and UNSIGNED mode from data group 1. Data group 1 contains one 32-bit register and two 64-bit registers.

```

UInt32 grp_size[2] = {1, 2}; // 1 register 32 bit, 2 registers 64 bit.
UInt32 reg_nr_32[1] = {6};    // 32 bit register no. 6 is in the group.
UInt32 reg_nr_64[2] = {4,5};  // 64 bit registers no. 4, 5 are also in the group.
UInt32* reg_nr[2];
reg_nr[0] = reg_nr_32;
reg_nr[1] = reg_nr_64;

dsfloat scaling_factor_32[1] = {IOFPGA_WRITE_SCALE_BIN_PT_0};
dsfloat scaling_factor_64[2] = {IOFPGA_WRITE_SCALE_BIN_PT_5, IOFPGA_WRITE_SCALE_BIN_PT_5};
dsfloat* scaling_factor[2];
scaling_factor[0] = scaling_factor_32;
scaling_factor[1] = scaling_factor_64;

UInt32 mode_32[1] = {IOFPGA_DATA_MODE_SIGNED};
UInt32 mode_64[2] = {IOFPGA_DATA_MODE_SIGNED, IOFPGA_DATA_MODE_UNSIGNED};
UInt32* mode[2];
mode[0] = mode_32;
mode[1] = mode_64;

dsfloat data_32[1] = {0};
dsfloat data_64[2] = {0, 0};
dsfloat* data[2];

IoFpga_read_reg_grp_mixed(
    1,
    grp_size,
    reg_nr,
    data,
    scaling_factor,
    mode);

```

Related topics**References**

IoFpga_init.....	890
IoFpga_read_reg.....	902
IoFpga_read_reg_grp.....	908
IoFpga_read_reg64.....	903
IoFpga_write_reg_grp_mixed.....	914

IoFpga_write_reg_grp

Syntax

```

__INLINE void IoFpga_write_reg_grp(
    UInt32 grp_nr,
    UInt32 grp_size,
    UInt32* reg_nr,
    Float64* data,
    Float64* scaling,
    UInt32* mode)

```

Include file	<code>IoFpga.h</code>																				
Purpose	To write to a group of exchange registers of the I/O FPGA application.																				
Description	This function writes data to several specified exchange registers and scales each value in the group by a specified factor. All the data in the group can be assumed to be consistent, which means that no single value can be accessed by the FPGA application until the last value of the group is written.																				
Parameters	<p>grp_nr Specifies the number of the data group in the range 1 ... 63.</p> <p>grp_size Specifies the group size. The number of values is in the range 1 ... 256.</p> <p>reg_nr Specifies the pointer to the variable containing the number of each group register in the range 1 ... 256. The variable itself is an array of grp_size size.</p> <p>Note For best performance, the registers in the group should be in ascending order without gaps.</p> <p>data Specifies the pointer to the variable containing the register values as an array of grp_size size.</p> <p>scaling Specifies the pointer to the variable containing the scaling factors as an array of grp_size size. Use the predefined symbols.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOFPGA_WRITE_SCALE_BIN_PT_0</code></td><td>Scaling factor: $2^0=1$</td></tr> <tr> <td><code>IOFPGA_WRITE_SCALE_BIN_PT_1</code></td><td>Scaling factor: $2^1=2$</td></tr> <tr> <td><code>IOFPGA_WRITE_SCALE_BIN_PT_2</code></td><td>Scaling factor: $2^2=4$</td></tr> <tr> <td><code>IOFPGA_WRITE_SCALE_BIN_PT_3</code></td><td>Scaling factor: $2^3=8$</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td><code>IOFPGA_WRITE_SCALE_BIN_PT_32</code></td><td>Scaling factor: $2^{32}=4,294,967,296$</td></tr> </tbody> </table> <p>mode Specifies the pointer to the variable containing the data modes as an array of grp_size size. Use the predefined symbols.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOFPGA_DATA_MODE_SIGNED</code></td><td>The data is processed as a signed data type.</td></tr> <tr> <td><code>IOFPGA_DATA_MODE_UNSIGNED</code></td><td>The data is processed as an unsigned data type.</td></tr> </tbody> </table>	Symbol	Meaning	<code>IOFPGA_WRITE_SCALE_BIN_PT_0</code>	Scaling factor: $2^0=1$	<code>IOFPGA_WRITE_SCALE_BIN_PT_1</code>	Scaling factor: $2^1=2$	<code>IOFPGA_WRITE_SCALE_BIN_PT_2</code>	Scaling factor: $2^2=4$	<code>IOFPGA_WRITE_SCALE_BIN_PT_3</code>	Scaling factor: $2^3=8$	<code>IOFPGA_WRITE_SCALE_BIN_PT_32</code>	Scaling factor: $2^{32}=4,294,967,296$	Symbol	Meaning	<code>IOFPGA_DATA_MODE_SIGNED</code>	The data is processed as a signed data type.	<code>IOFPGA_DATA_MODE_UNSIGNED</code>	The data is processed as an unsigned data type.
Symbol	Meaning																				
<code>IOFPGA_WRITE_SCALE_BIN_PT_0</code>	Scaling factor: $2^0=1$																				
<code>IOFPGA_WRITE_SCALE_BIN_PT_1</code>	Scaling factor: $2^1=2$																				
<code>IOFPGA_WRITE_SCALE_BIN_PT_2</code>	Scaling factor: $2^2=4$																				
<code>IOFPGA_WRITE_SCALE_BIN_PT_3</code>	Scaling factor: $2^3=8$																				
...	...																				
<code>IOFPGA_WRITE_SCALE_BIN_PT_32</code>	Scaling factor: $2^{32}=4,294,967,296$																				
Symbol	Meaning																				
<code>IOFPGA_DATA_MODE_SIGNED</code>	The data is processed as a signed data type.																				
<code>IOFPGA_DATA_MODE_UNSIGNED</code>	The data is processed as an unsigned data type.																				

Symbol	Meaning
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 24, which complies with IEEE 754 standard (single). The scaling factor is ignored in this mode.

Return value None

Example This example shows you how to write data in signed mode to data group 1. It contains 1 value that is scaled by 2^{16} :

```
Float64 scaling_factor = IOFPGA_WRITE_SCALE_BIN_PT_16;
UInt32 mode = IOFPGA_DATA_MODE_SIGNED;
Float64 data = 1.3;
...
IoFpga_write_reg_grp(
    1,
    1,
    &reg_nr,
    &data,
    &scaling_factor,
    &mode);
```

Related topics References

IoFpga_init.....	890
IoFpga_read_buf.....	917
IoFpga_read_reg.....	902
IoFpga_read_reg_grp.....	908
IoFpga_write_buf.....	921
IoFpga_write_reg.....	905

IoFpga_write_reg_grp_mixed

Syntax

```
__INLINE void IoFpga_write_reg_grp_mixed(
    UInt32 grp_nr,
    UInt32 grp_size[2],
    UInt32* reg_nr[2],
    Float64* data[2],
    Float64* scaling[2],
    UInt32* mode[2])
```

Include file	<code>IoFpga.h</code>																										
Purpose	To write to a group of 32-bit and 64-bit exchange registers of the I/O FPGA application.																										
Description	This function writes data to several specified exchange registers and scales each value in the group by a specified factor. All the data in the group can be assumed to be consistent, which means that no single value can be accessed by the FPGA application until the last value of the group is written.																										
Parameters	<p>grp_nr Specifies the number of the data group in the range 1 ... 63.</p> <p>grp_size Specifies the number of 32-bit and 64-bit registers in the group. The number of values is in the range 1 ... 256.</p> <table border="1"> <thead> <tr> <th>Item</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>grp_size[0]</code></td><td>Specifies the number of 32-bit registers in the group.</td></tr> <tr> <td><code>grp_size[1]</code></td><td>Specifies the number of 64-bit registers in the group.</td></tr> </tbody> </table> <p>reg_nr Specifies the pointers to the variables containing the number of each group register in the range 1 ... 256. The variable itself is an array of grp_size size.</p> <p>Note For best performance, the registers in the group should be in ascending order without gaps.</p> <table border="1"> <thead> <tr> <th>Item</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>reg_nr[0]</code></td><td>Specifies the register numbers of the 32-bit registers in the group.</td></tr> <tr> <td><code>reg_nr[1]</code></td><td>Specifies the register numbers of the 64-bit register in the group.</td></tr> </tbody> </table> <p>data Specifies the pointers to the variables containing the register values as an array of grp_size size.</p> <table border="1"> <thead> <tr> <th>Item</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>data[0]</code></td><td>Specifies the register data of the 32-bit registers in the group.</td></tr> <tr> <td><code>data[1]</code></td><td>Specifies the register data of the 64-bit registers in the group.</td></tr> </tbody> </table> <p>scaling Specifies the pointers to the variables containing the scaling factors as an array of grp_size size. Use the predefined symbols.</p> <table border="1"> <thead> <tr> <th>Symbol</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td><code>IOFPGA_READ_SCALE_BIN_PT_0</code></td><td>Scaling factor: $1/2^0=1.00$</td></tr> <tr> <td><code>IOFPGA_READ_SCALE_BIN_PT_1</code></td><td>Scaling factor: $1/2^1=0.50$</td></tr> <tr> <td><code>IOFPGA_READ_SCALE_BIN_PT_2</code></td><td>Scaling factor: $1/2^2=0.25$</td></tr> </tbody> </table>	Item	Meaning	<code>grp_size[0]</code>	Specifies the number of 32-bit registers in the group.	<code>grp_size[1]</code>	Specifies the number of 64-bit registers in the group.	Item	Meaning	<code>reg_nr[0]</code>	Specifies the register numbers of the 32-bit registers in the group.	<code>reg_nr[1]</code>	Specifies the register numbers of the 64-bit register in the group.	Item	Meaning	<code>data[0]</code>	Specifies the register data of the 32-bit registers in the group.	<code>data[1]</code>	Specifies the register data of the 64-bit registers in the group.	Symbol	Meaning	<code>IOFPGA_READ_SCALE_BIN_PT_0</code>	Scaling factor: $1/2^0=1.00$	<code>IOFPGA_READ_SCALE_BIN_PT_1</code>	Scaling factor: $1/2^1=0.50$	<code>IOFPGA_READ_SCALE_BIN_PT_2</code>	Scaling factor: $1/2^2=0.25$
Item	Meaning																										
<code>grp_size[0]</code>	Specifies the number of 32-bit registers in the group.																										
<code>grp_size[1]</code>	Specifies the number of 64-bit registers in the group.																										
Item	Meaning																										
<code>reg_nr[0]</code>	Specifies the register numbers of the 32-bit registers in the group.																										
<code>reg_nr[1]</code>	Specifies the register numbers of the 64-bit register in the group.																										
Item	Meaning																										
<code>data[0]</code>	Specifies the register data of the 32-bit registers in the group.																										
<code>data[1]</code>	Specifies the register data of the 64-bit registers in the group.																										
Symbol	Meaning																										
<code>IOFPGA_READ_SCALE_BIN_PT_0</code>	Scaling factor: $1/2^0=1.00$																										
<code>IOFPGA_READ_SCALE_BIN_PT_1</code>	Scaling factor: $1/2^1=0.50$																										
<code>IOFPGA_READ_SCALE_BIN_PT_2</code>	Scaling factor: $1/2^2=0.25$																										

Symbol	Meaning
IOFGA_READ_SCALE_BIN_PT_3	Scaling factor: $1/2^3=0.125$
...	...
IOFGA_READ_SCALE_BIN_PT_64	Scaling factor: $1/2^{64}=5.421e^{-20}$

Item	Meaning
scaling[0]	Specifies the scaling of each 32-bit register in the group.
scaling[1]	Specifies the scaling of each 64-bit register in the group.

mode Specifies the pointers to the variables containing the data modes as an array of **grp_size** size. Use the predefined symbols.

Symbol	Meaning
IOFGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 24 or 53, which complies with IEEE 754 standard (single or double). The scaling factor is ignored in this mode.

Item	Meaning
mode[0]	Specifies the mode of each 32-bit register in the group.
mode[1]	Specifies the mode of each 64-bit register in the group.

Return value None

Example This example shows how to write data in SIGNED and UNSIGNED mode to data group 1. Data group 1 contains one 32-bit register and two 64-bit registers.

```
UInt32 grp_size[2] = {1, 2}; // 1 register 32 bit, 2 registers 64 bit.
UInt32 reg_nr_32[1] = {6}; // 32 bit register no. 6 is in the group.
UInt32 reg_nr_64[2] = {4,5}; // 64 bit registers no. 4, 5 are also in the group.
UInt32* reg_nr[2];
reg_nr[0] = reg_nr_32;
reg_nr[1] = reg_nr_64;

dsfloat scaling_factor_32[1] = {IOFGA_WRITE_SCALE_BIN_PT_0};
dsfloat scaling_factor_64[2] = {IOFGA_WRITE_SCALE_BIN_PT_5, IOFGA_WRITE_SCALE_BIN_PT_5};
dsfloat* scaling_factor[2];
scaling_factor[0] = scaling_factor_32;
scaling_factor[1] = scaling_factor_64;
```

```

UInt32 mode_32[1] = {IOFPGA_DATA_MODE_SIGNED};
UInt32 mode_64[2] = {IOFPGA_DATA_MODE_SIGNED, IOFPGA_DATA_MODE_UNSIGNED};
UInt32* mode[2];
mode[0] = mode_32;
mode[1] = mode_64;

dsfloat data_32[1] = {1.3};
dsfloat data_64[2] = {-2.5, 4.0};
dsfloat* data[2];
data[0] = data_32;
data[1] = data_64;

IoFpga_write_reg_grp_mixed(
    1,
    grp_size,
    reg_nr,
    data,
    scaling_factor,
    mode);

```

Related topics**References**

IoFpga_init.....	890
IoFpga_read_reg_grp_mixed.....	910
IoFpga_write_reg.....	905
IoFpga_write_reg_grp.....	912
IoFpga_write_reg64.....	906

IoFpga_read_buf

Syntax

```

__INLINE void IoFpga_read_buf(
    UInt32 buf_nr,
    UInt32* data_length,
    Float64* data,
    Float64* scaling,
    UInt32 mode,
    UInt32* status)

```

Include file

IoFpga.h

Purpose

To read data from a buffer of the I/O FPGA application.

Description

This function reads a specified number of values from a buffer and scales the data by a specified factor. You can configure the scaling and the mode parameters only for the entire buffer and not separately for each buffer value. The buffer state is also returned.

Parameters	<p>buf_nr Specifies the number of the buffer in the range 1 ... 32.</p> <p>data_length Specifies the pointer to the variable containing the data length as the number of buffer elements in the range 1 ... buffer_size. The maximum buffer size depends on the buffer definition in the FPGA framework used.</p> <p>data Specifies the pointer to the variable containing the buffer values as an array of data_length size.</p> <p>scaling Specifies the pointer to the variable containing the scaling factor for reading. Use the predefined symbols.</p>
-------------------	---

Symbol	Meaning
IOFPGA_READ_SCALE_BIN_PT_0	Scaling factor: $1/2^0 = 1.00$
IOFPGA_READ_SCALE_BIN_PT_1	Scaling factor: $1/2^1 = 0.50$
IOFPGA_READ_SCALE_BIN_PT_2	Scaling factor: $1/2^2 = 0.25$
IOFPGA_READ_SCALE_BIN_PT_3	Scaling factor: $1/2^3 = 0.125$
...	...
IOFPGA_READ_SCALE_BIN_PT_32	Scaling factor: $1/2^{32} = 2.328e^{-10}$

mode Specifies the data processing mode. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFPGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 24, which complies with IEEE 754 standard (single). The scaling factor is ignored in this mode.

status Specifies the pointer to the variable containing the buffer status. To check the buffer status, use the predefined symbols.

Symbol	Meaning
IOFPGA_BUF_OVFL	Data loss due to buffer overflow.
IOFPGA_BUF_NEW	Data is new, buffer was not read before.

Return value	None
---------------------	------

Example

This example shows you how to read data in signed mode from a buffer. The values are scaled by $1/2^{16}$:

```
Float64 data[1024];
UInt32 data_length;
Float64 scaling_factor = IOFPGA_READ_SCALE_BIN_PT_16;
UInt32 status;
...
IoFpga_read_buf(
    1,
    &data_length,
    data,
    &scaling_factor,
    IOFPGA_DATA_MODE_SIGNED,
    &status);
```

Related topics**References**

IoFpga_init.....	890
IoFpga_read_reg.....	902
IoFpga_read_reg_grp.....	908
IoFpga_write_buf.....	921
IoFpga_write_reg.....	905
IoFpga_write_reg_grp.....	912

IoFpga_read_buf64

Syntax

```
__INLINE void IoFpga_read_buf64(
    UInt32 buf_nr,
    UInt32* data_length,
    Float64* data,
    Float64* scaling,
    UInt32 mode,
    UInt32* status)
```

Include file

IoFpga.h

Purpose

To read 64-bit data from a buffer of the I/O FPGA application.

Description

This function reads a specified number of values from a buffer and scales the 64-bit data by a specified factor. You can configure the scaling and the mode parameters only for the entire buffer and not separately for each buffer value. The buffer state is also returned.

Parameters	<p>buf_nr Specifies the number of the buffer in the range 1 ... 32.</p> <p>data_length Specifies the pointer to the variable containing the data length as the number of buffer elements in the range 1 ... buffer_size. The maximum buffer size depends on the buffer definition in the FPGA framework used.</p> <p>data Specifies the pointer to the variable containing the buffer values as an array of data_length size.</p> <p>scaling Specifies the pointer to the variable containing the scaling factor for reading. Use the predefined symbols.</p>
-------------------	---

Symbol	Meaning
IOFPGA_READ_SCALE_BIN_PT_0	Scaling factor: $1/2^0=1.00$
IOFPGA_READ_SCALE_BIN_PT_1	Scaling factor: $1/2^1=0.50$
IOFPGA_READ_SCALE_BIN_PT_2	Scaling factor: $1/2^2=0.25$
IOFPGA_READ_SCALE_BIN_PT_3	Scaling factor: $1/2^3=0.125$
...	...
IOFPGA_READ_SCALE_BIN_PT_64	Scaling factor: $1/2^{64}=5.421e^{-20}$

mode Specifies the data processing mode. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFPGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 53, which complies with IEEE 754 standard (double). The scaling factor is ignored in this mode.

status Specifies the pointer to the variable containing the buffer status. To check the buffer status, use the predefined symbols.

Symbol	Meaning
IOFPGA_BUF_OVFL	Data loss due to buffer overflow.
IOFPGA_BUF_NEW	Data is new, buffer was not read before.

Return value	None
---------------------	------

Example

This example shows you how to read data in signed mode from a buffer. The values are scaled by $1/2^{16}$:

```
Float64 data[1024];
UInt32 data_length;
Float64 scaling_factor = IOFPGA_READ_SCALE_BIN_PT_16;
UInt32 status;
...
IoFpga_read_buf64(
    1,
    &data_length,
    data,
    &scaling_factor,
    IOFPGA_DATA_MODE_SIGNED,
    &status);
```

Related topics**References**

IoFpga_init	890
IoFpga_read_buf	917
IoFpga_read_reg64	903
IoFpga_write_buf64	923

IoFpga_write_buf

Syntax

```
__INLINE void IoFpga_write_buf(
    UInt32 buf_nr,
    UInt32 data_length,
    Float64* data,
    Float64* scaling,
    UInt32 mode)
```

Include file**IoFpga.h****Purpose**

To write data to a buffer of the I/O FPGA application.

Description

This function writes a specified number of values to a buffer and scales the data by a specified factor.

Parameters	<p>buf_nr Specifies the number of the buffer in the range 1 ... 32.</p> <p>data_length Specifies the data length as the number of buffer elements in the range 1 ... buffer_size. The maximum buffer size depends on the buffer definition in the FPGA framework used.</p> <p>data Specifies the pointer to the variable containing the buffer values as an array of data_length size.</p> <p>scaling Specifies the pointer to the variable containing the scaling factor for writing. Use the predefined symbols.</p>
-------------------	--

Symbol	Meaning
IOFPGA_WRITE_SCALE_BIN_PT_0	Scaling factor: $2^0=1$
IOFPGA_WRITE_SCALE_BIN_PT_1	Scaling factor: $2^1=2$
IOFPGA_WRITE_SCALE_BIN_PT_2	Scaling factor: $2^2=4$
IOFPGA_WRITE_SCALE_BIN_PT_3	Scaling factor: $2^3=8$
...	...
IOFPGA_WRITE_SCALE_BIN_PT_32	Scaling factor: $2^{32}=4,294,967,296$

mode Specifies the data processing mode. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFPGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 24, which complies with IEEE 754 standard (single). The scaling factor is ignored in this mode.

Return value	None
---------------------	------

Example This example shows you how to write 128 values in signed mode to a buffer. The values are scaled by 2^{16} :

```
Float64 data[128] = {1.3};
Float64 scaling_factor = IOFPGA_WRITE_SCALE_BIN_PT_16;
...
IoFpga_write_buf(
    1,
    128,
    &data,
    &scaling_factor,
    IOFPGA_DATA_MODE_SIGNED);
```

Related topics

References

IoFpga_init.....	890
IoFpga_read_buf.....	917
IoFpga_read_reg.....	902
IoFpga_read_reg_grp.....	908
IoFpga_write_reg.....	905
IoFpga_write_reg_grp.....	912

IoFpga_write_buf64

Syntax

```
__INLINE void IoFpga_write_buf64(
    UInt32 buf_nr,
    UInt32 data_length,
    Float64* data,
    Float64* scaling,
    UInt32 mode
```

Include file

IoFpga.h

Purpose

To write 64-bit data to a buffer of the I/O FPGA application.

Description

This function writes a specified number of values to a buffer and scales the 64-bit data by a specified factor.

Parameters**buf_nr** Specifies the number of the buffer in the range 1 ... 32.**data_length** Specifies the data length as the number of buffer elements in the range 1 ... **buffer_size**. The maximum buffer size depends on the buffer definition in the FPGA framework used.**data** Specifies the pointer to the variable containing the buffer values as an array of **data_length** size.**scaling** Specifies the pointer to the variable containing the scaling factor for writing. Use the predefined symbols.

Symbol	Meaning
IOFPGA_WRITE_SCALE_BIN_PT_0	Scaling factor: $2^0=1$
IOFPGA_WRITE_SCALE_BIN_PT_1	Scaling factor: $2^1=2$
IOFPGA_WRITE_SCALE_BIN_PT_2	Scaling factor: $2^2=4$
IOFPGA_WRITE_SCALE_BIN_PT_3	Scaling factor: $2^3=8$

Symbol	Meaning
...	...
IOFPGA_WRITE_SCALE_BIN_PT_64	Scaling factor: $2^{64}=18,446,744,073,709,551,616$

mode Specifies the data processing mode. Use one of the predefined symbols.

Symbol	Meaning
IOFPGA_DATA_MODE_SIGNED	The data is processed as a signed data type.
IOFPGA_DATA_MODE_UNSIGNED	The data is processed as an unsigned data type.
IOFPGA_DATA_MODE_FLOAT	The data is processed as floating point with a fraction width of 53, which complies with IEEE 754 standard (double). The scaling factor is ignored in this mode.

Return value	None
---------------------	------

Example	This example shows you how to write 128 values in signed mode to a buffer. The values are scaled by 2^{16} :
----------------	--

```
Float64 data[128] = {1.3};
Float64 scaling_factor = IOFPGA_WRITE_SCALE_BIN_PT_16;
...
IoFpga_write_buf64(
    1,
    128,
    &data,
    &scaling_factor,
    IOFPGA_DATA_MODE_SIGNED);
```

Related topics	References
-----------------------	-------------------

IoFpga_init.....	890
IoFpga_read_buf64.....	919
IoFpga_write_buf.....	921
IoFpga_write_reg64.....	906

Host Programs

Introduction There are some utilities installed on the host PC for building custom applications.

Where to go from here	Information in this section
	<p>Host Settings.....926 For information about the necessary settings of the software environment and the DS1202 Real-Time Library.</p> <p>Compiling, Linking and Downloading an Application.....929 Information about the batch files, makefiles, and linker command files that support your program development.</p> <p>Debugging an Application.....937 Information about disassembling via <code>ntoppc-objdump</code>.</p>

Information in other sections

[Firmware Manager Manual](#)

Introduces you to the features provided by the Firmware Manager. It provides detailed information on the user interface, its command line options and instructions using the firmware management.

Host Settings

Introduction

This chapter describes the definitions, settings, files and libraries that are necessary to write your own C-coded programs for the PowerPC processor of MicroLabBox.

Where to go from here

Information in this section

Compiler and C Run-Time Libraries.....	926
Environment Variables and Paths.....	926
Folder Structure.....	927
DS1202 Real-Time Library.....	927
File Extensions.....	927

Compiler and C Run-Time Libraries

Compiler and C run-time libraries

The compiler for building MicroLabBox applications is automatically installed when you install dSPACE software. The associated C run-time libraries are also used. The GNU compiler for QNX is installed in `<RCP_HIL_InstallationPath>\Compiler`. Further GNU tools, with the prefix `n toppc` are installed for MicroLabBox.

For information on the C++ support, refer to [Integrating C++ Code](#) on page 935.

Environment Variables and Paths

dSPACE command prompt

The dSPACE software installation does not set environment variables and other settings such as enhancements to the search path.

Use the Command Prompt for dSPACE RCP and HIL for the host tools. You find the command prompt as a shortcut in the Windows Start menu. The required paths and environment settings are then automatically set.

Folder Structure

Folder structure

The folder structure of the DS1202 software is as follows:

Folder	Contents
<RCP_HIL_InstallationPath>\DS1202\Lib	Library files of MicroLabBox (DS1202)
<RCP_HIL_InstallationPath>\DS1202\Include	Header files of MicroLabBox
<RCP_HIL_InstallationPath>\DS1202	Makefiles of MicroLabBox
<RCP_HIL_InstallationPath>\DS1202\Win32	MicroLabBox related host tools and DLLs.
<RCP_HIL_InstallationPath>\Demos\DS1202	Demo examples

DS1202 Real-Time Library

MicroLabBox libraries

All functions of the DS1202 Real-Time Library were compiled with the highest optimization level. Required objects from this library are dynamically linked to an application. This lets you update the RTLib1202 without having to rebuild the application. The header files are located in <RCP_HIL_InstallationPath>DS1202\Include.

Note

All necessary modules and header files are included by Brtenv.h.

File Extensions

File extensions

The following file extensions are used:

File Extension	Meaning
.c ¹⁾	C source files
.cpp	C++ source files
.so	Dynamically linked libraries (shared objects)
.a	Statically linked libraries (archives)
.mk	Makefiles
.ppc	Executable programs for the PowerPC

File Extension	Meaning
.rta	Real-time application file, contains the PPC file and further required files, if available.

¹⁾ For C++ support, refer to [Integrating C++ Code](#) on page 935.

Compiling, Linking and Downloading an Application

Introduction

If you want to build a user application and download it to the target hardware, you can use the **Down** tool for your board.

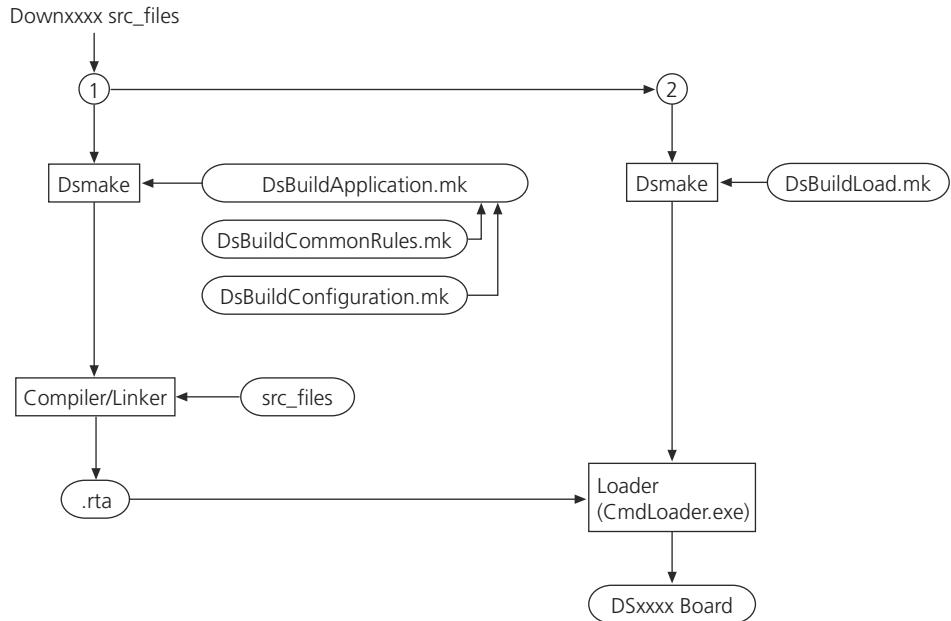
Tip

The executable file Down1202 can be called in a Command Prompt window (DOS window) of your host PC.

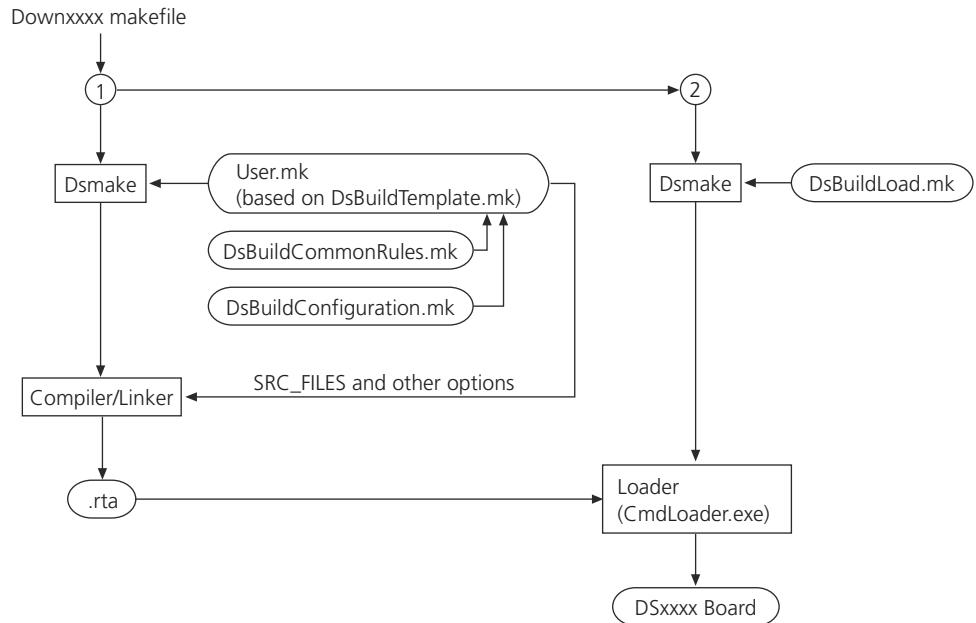
If you use the **Command Prompt for dSPACE RCP and HIL** shortcut in the Windows Start menu, the required paths and environment settings are automatically set.

Process overview

The following schematic shows you the process overview for using Down with the source files as arguments. **DsBuildApplication.mk** is then used for the make process.



The following schematic shows you the process overview for using Down with the custom makefile as an argument. It is recommended to base the makefile on **DsBuildTemplate.mk**.

**Where to go from here****Information in this section**

Down1202.exe	930
To compile, link, and download applications.	
DsBuildApplication.mk	933
This is the default makefile if you use Down with source files as arguments.	
DsBuildLoad.mk	934
To download an application to the target hardware.	
DsBuildTemplate.mk	934
This is a template for a custom makefile.	
Integrating C++ Code	935
Gives you instructions on enabling the C++ support.	

Down1202.exe

Syntax

```
down1202 file.mk [options] [/?]
```

or

```
down1202 src_file(s) [options] [/?]
```

Purpose	To compile or assemble, link, and download handcoded applications.
Description	<p>The following file types can be handled:</p> <p>Local makefile (.mk) To compile, link, and download the application using the specified local makefile. Use the makefile DsBuildTemplate.mk as a template to write your own makefile. The resulting program file is named according to the name of the specified makefile.</p> <p>C-coded source file (.c) To specify the file(s) to be compiled and linked using DsBuildApplication.mk. The resulting program file is named according to the name of the first specified source file.</p> <p>C++-coded source file (.cpp) To specify the file(s) to be compiled and linked using DsBuildApplication.mk. The resulting program file is named according to the name of the first specified source file.</p>
	<div style="border: 1px solid #ccc; padding: 5px;"> <p>Note</p> <p>If you use the Down tool with source files, the relocatable object files are deleted and the object file of the application is overwritten. If you call the Down tool with a user makefile as the argument, the object files remain unchanged until a modified source file requires recompilation.</p> </div>
	<p>If the file name extension is omitted, Down1202 searches for existing files in the above order. If more than one source file is specified at the command line, the first file is treated as the main source file that names the complete application. The remaining source files are compiled or assembled, and linked to the application.</p> <p>The built application is loaded by default to the DS1202 platform named 'ds1202'. The platform name is set by the dSPACE software, e.g., ControlDesk during platform registration. If you want to access another platform instead, you can specify the platform name using the /p option.</p> <p>For a graphical process overview, refer to Compiling, Linking and Downloading an Application on page 929.</p> <p>For further information on the C++ support, refer to Integrating C++ Code on page 935.</p>

Options	The following command line options are available:
Option	Meaning
/c <NetworkName>	To specify the platform to be registered and used.
/co <option>	To specify additional compiler options; refer to the GNU Compiler documentation.
/d	To disable downloading the application; only compiling and linking.
/g	To compile for source level debugging;
/l	To write all output to down1202.log .

Option	Meaning
/lib <lib_file>	To specify an additional library to be linked.
/lko <option>	To specify a single additional linker option.
/lo <option>	To specify a single additional loader option.
/mo <option>	To specify a single additional DSMAKE option; call <code>dsmake -h</code> to get more information.
/n	To disable beep on error.
/p <PlatformName>	To specify a platform name that differs from the default. The default platform name is ds1202 if you call <code>Down1202.exe</code> .
/pause	To pause execution of <code>Down1202</code> before exit.
/x	To switch off code optimization.
/z	To download an existing object file without building.
/?	To display information.

Messages

The following messages are defined:

Message	Description
ERROR: not enough memory!	The attempt to allocate dynamic memory failed.
ERROR: environment variable PPC_ROOT not found! ERROR: environment variable X86_ROOT not found! ERROR: environment variable DSPACE_ROOT not found!	The respective environment variable is not defined in the DOS environment. The environment variables are set during the dSPACE software installation.
ERROR: can't load DLL '%DSPACE_ROOT %/exe/wbinfo.dll'! [number]	Loading the dynamic link library WBINFO.DLL failed. The number in brackets specifies the internal Windows error.
ERROR: can't read address of function 'GetWorkingBoardName()'! ERROR: can't read address of function 'GetWorkingBoardClient()'! ERROR: can't read address of function 'GetWorkingBoardConnection()'! ERROR: can't read address of function 'GetWorkingBoardType()'!	The address of the respective function could not be found in the dynamic link library WBINFO.DLL.
ERROR: can't read working board name! ERROR: can't read working board client! ERROR: can't read working board connection! ERROR: can't read working board type!	The respective working board information could not be read from the dspace.ini file. Register your hardware system using ControlDesk's Platform Manager.
WARNING: The working board type is DS???? instead of DS????! Accessing default board ds????.	The detected working board type is not responding to the DOWN1202 version. For example, if you are using DOWN1202 and the working board is of type DS1104, the board name ds1202 is used.
ERROR: unable to obtain full path of <file name>!	This error occurs if the full path name of the source file contains more than 260 characters, or if an invalid drive letter has been specified, for example, 1:\test.

Message	Description
ERROR: unable to access file <file name>!	The specified file cannot be accessed by Down1202. The file does not exist or another application is accessing it.
ERROR: source files must be available in the same directory!	All source files to be compiled must be available in the same application folder.
ERROR: make file <name> not allowed as additional source file!	Only assembly and C source files are allowed as additional source files.
ERROR: can't redirect stdout to file! ERROR: can't redirect stdout to screen!	The redirection of stdout to a file or to the screen has failed.
ERROR: can't invoke %DSpace_ROOT %\exe\dsmake.exe: ...	Down1202 was not able to invoke DSMAKE.EXE successfully.
ERROR: making of <file name> failed! ERROR: building of <file name> failed!	An error occurred while executing a makefile, compiling or assembling a source file. See the screen output to get information about the reasons, for example, there can be programming errors in the source file.
ERROR: downloading of <file name> failed!	DOWN1202 was not able to download the application successfully. See dSPACE.log for more information.
ERROR: can't install exit handler!	The available memory space is too small for registering the exit handler.

Related topics**References**

DsBuildApplication.mk.....	933
DsBuildLoad.mk.....	934
DsBuildTemplate.mk....	934

DsBuildApplication.mk

Description

This makefile is used to compile or assemble the application source files. It is called by Down1202.exe if no other makefile is specified. It uses the highest optimization level of the C compiler.

It includes:

- DsBuildCommonRules.mk
- DsBuildConfiguration.mk

Note

Do not edit.

Use the required option with Down1202 or a custom makefile based on DsBuildTemplate.mk instead.

Related topics

References

Down1202.exe.....	930
DsBuildTemplate.mk.....	934

DsBuildLoad.mk

Description

This file is automatically invoked by Down1202.exe to load the application to the target hardware after building, unless you use the /d option. It is also called if you use the /z option for download only.

Note

Do not edit or change this file.

Related topics

References

Down1202.exe.....	930
-------------------	-----

DsBuildTemplate.mk

Description

You can customize this makefile to match your individual requirements:

- CUSTOM_SRC_FILES

You can add additional source files to be compiled by adding the names of the source files.

- CUSTOM_OBJ_FILES

You can add additional object files to be linked to the application by adding the names of the object files.

- **CUSTOM_LIB_FILES**
You can add additional libraries to be linked to the application by adding the names of the libraries.
- **CUSTOM_C_OPTS**
You can add additional options for the C compiler.
- **CUSTOM_ASM_OPTS**
You can add additional options for the assembler.
- **CUSTOM_LK_OPTS**
You can add additional options for the linker.
- **USER_BUILD_CPP_APPL**
You can enable the C++ support by setting this make macro to ON. For further information, refer to [Integrating C++ Code](#) on page 935.

Related topics**References**

Down1202.exe.....	930
-------------------	-----

Integrating C++ Code

Introduction

To integrate C++ code to your handcoded RTLib application, you have to enable the C++ support.

Adapting the user makefile

For adding C++ code to your application you have to adapt the `DsBuildTemplate.mk` file.

- Enable the C++ support
- Add C++ source files, C++ object files and C++ libraries

Example:

```
# Enable C++ support
USER_BUILD_CPP_APPL = ON
...
# Additional C/C++ source files to be compiled
CUSTOM_SRC_FILES = main.c example.cpp
...
# Additional user object files to be linked
USER_OBJS = MyModule3.o03 MyModule4.cppo03
...
# Additional user libraries to be linked
USER_LIBS = MyCLib.lib MyCppLib.lib
```

The number at the object files reflect the different processor platforms. In the example, `.o03` and `.cppo03` shows that the files has been built for a DS1401 (MicroAutoBox II).

For further information on the user makefile, refer to [DsBuildTemplate.mk](#) on page 934.

Debugging an Application

Introduction

Simple application errors can be found by implementing messages in your source code to log measured or calculated values of variables (refer to [Message Handling](#) on page 130).

ntoppc-objdump

Syntax

```
ntoppc-objdump [options] objfile
```

Purpose

To display information about one or more object files.

Description

This utility is mainly used for debugging purposes. For example, it can disassemble an object file and show the machine instructions with their memory locations. The display of particular information is controlled by command line options. At least one option besides **-l** (**--line-numbers**) must be specified.

Note

To make it possible for ntoppc-objdump to display correlating source code information, you must build your application with the debug option **-g**.

You can find this utility in
`<RCP_HIL_InstallationPath>\Compiler\QNX650\host\win32\x86\usr\b
in.`

Options

The following command line options are available:

Option	Meaning
-a	--archive-headers Shows object file format and header information from an archive object file.
	--adjust-vma=<offset> Adds offset to all the section addresses. This is useful for dumping information, if the section addresses do not correspond to the symbol table.
-b <bfdname>	--target=<bfdname> Specifies the object code format as bfdname . This might not be necessary, because many formats can be recognized automatically. You can list the available formats with -i .
-g	--debugging Displays debugging information using a C-like syntax.
-e	--debugging-tags Displays debugging information using ctags style.

Option	Meaning
<code>-W</code>	--dwarf
<code>-C</code>	--demangle
	As style, you can specify:
	<ul style="list-style-type: none"> ▪ auto ▪ gnu ▪ lucid ▪ arm ▪ hp ▪ edg ▪ gnu-v3 ▪ java ▪ gnat
<code>-d</code>	--disassemble
<code>-D</code>	--disassemble-all
<code>-M</code>	--disassembler-options=<options>
<code>-z</code>	--disassemble-zeroes
	--prefix-addresses
<code>-EB</code>	--endian=big
<code>-EL</code>	--endian=little
<code>-f</code>	--file-headers
<code>-h</code>	--section-headers
	--headers
<code>-H</code>	--help
@<file>	
<code>-i</code>	--info
<code>-I</code>	--include=<folder>
<code>-j <section></code>	--section=<section>
<code>-l</code>	--line-numbers
<code>-m <machine></code>	--architecture=<machine>
<code>-p</code>	--private-headers
<code>-r</code>	--reloc

Option	Meaning
-R	--dynamic-reloc Displays the dynamic relocation entries of the object file. This is only useful for dynamic objects, such as certain types of shared libraries.
-s	--full-contents Displays the full contents of any sections requested.
-S	--source Displays source code intermixed with disassembly, if possible. This option implies -d.
	--show-rawInsn Displays disassembled instructions in HEX as well as in symbolic form.
	--no-show-rawInsn Does not display the instruction bytes of disassembled instructions.
-G	--stabs Displays the contents of .stab, .stab.index and .stab.excl sections from an ELF file.
	--start-address=<address> Starts displaying at the specified address. This affects the output of the -d, -r and -s options.
	--stop-address=<address> Stops displaying at the specified address. This affects the output of the -d, -r and -s options.
-t	--syms Displays the symbol table entries of the object file.
-T	--dynamic-syms Displays the dynamic symbol table entries of the object file. This is only useful for dynamic objects, such as certain types of shared libraries.
-w	--wide Formats some lines for output devices that have more than 80 columns.
-v	--version Displays the version number.
-x	--all-headers Displays all available header information, including the symbol table and relocation entries. This option implies -a, -f, -h, -r and -t.

Example

For debugging an application, it is useful to disassemble all sections together with information on the line numbers and corresponding source code of the displayed assembler instructions. ntoppc-objdump prints a great amount of data, so it is recommended to redirect the output to a dump file, which you can open with a text editor. The command looks like this:

```
ntoppc-objdump -S -l -D appl.rta > result.dmp
```


A

A/D conversion 202
 Analog In Class 1 202
 Analog In Class 2 250
 access functions
 CAN 803
 ACU 493
AdcCl1AnalogIn_abortBurst 204
AdcCl1AnalogIn_abortBurstMultiple 205
AdcCl1AnalogIn_apply 206
AdcCl1AnalogIn_create 208
AdcCl1AnalogIn_disableInterrupts 209
AdcCl1AnalogIn_enableInterrupts 211
AdcCl1AnalogIn_getBurstCurrent 212
AdcCl1AnalogIn_getBurstImmediate 214
AdcCl1AnalogIn_getFailureInfo 216
AdcCl1AnalogIn_getSingleValue 218
AdcCl1AnalogIn_isDataReady 219
AdcCl1AnalogIn_read 221
AdcCl1AnalogIn_readFailureInfo 222
AdcCl1AnalogIn_setBurstMode 223
AdcCl1AnalogIn_setBurstSize 225
AdcCl1AnalogIn_setBurstTrigger 226
AdcCl1AnalogIn_setChannelState 228
AdcCl1AnalogIn_setConversionMode 229
AdcCl1AnalogIn_setConversTrigger 231
AdcCl1AnalogIn_setInterruptMode 233
AdcCl1AnalogIn_setIoIntVector 234
AdcCl1AnalogIn_setTimerPeriod 236
AdcCl1AnalogIn_setTriggerLineIn 237
AdcCl1AnalogIn_setTriggerLineOut 238
AdcCl1AnalogIn_setTriggerLineOutStatus 240
AdcCl1AnalogIn_setTriggerMode 241
AdcCl1AnalogIn_start 243
AdcCl1AnalogIn_stop 244
AdcCl1AnalogIn_triggerBurst 245
AdcCl1AnalogIn_triggerBurstMultiple 246
AdcCl1AnalogIn_triggerConversion 247
AdcCl1AnalogIn_triggerConversionMultiple 248
AdcCl1AnalogIn_write 249
AdcCl2AnalogIn_apply 251
AdcCl2AnalogIn_create 252
AdcCl2AnalogIn_getInputValue 254
AdcCl2AnalogIn_read 255
AdcCl2AnalogIn_start 256
 advanced I/O functions
 electric motor control 491
 Analog In Class 1 202
 Analog In Class 2 250
 Analog Out Class 1 259
 angle computation unit 493
 application
 debugging 937

B

basic functions 25
 Bit I/O 268
Digital In Class 1 268

Digital Out Class 1 290
 block-commutated PWM generation
 class 1 725
 buzzer control 153
Buzzer_apply 154
Buzzer_create 155
Buzzer_setDuration 156
Buzzer_setFrequency 157
Buzzer_setNumberOfBeeps 158
Buzzer_setPause 160
Buzzer_start 161
Buzzer_write 161

C

CAN
 access functions 803
 basic communication principles 804
 communication channel 804
 initialization functions 804
 read functions 804
 register functions 804
 request functions 804
 slave access functions 804
 write functions 804
can_tp1_all_data_clear 877
can_tp1_can_msg_send_id 853
can_tp1_can_msg_set 847
can_tp1_canChannel 807
can_tp1_canMsg 812
can_tp1_canService 809
can_tp1_channel_all_sleep 822
can_tp1_channel_all_wakeup 823
can_tp1_channel_BOff_go 824
can_tp1_channel_BOff_return 825
can_tp1_channel_init 818
can_tp1_channel_set 826
can_tp1_channel_start 821
can_tp1_channel_txqueue_clear 828
can_tp1_communication_init 816
can_tp1_error_read 878
can_tp1_msg_clear 865
can_tp1_msg_processed_read 868
can_tp1_msg_processed_register 866
can_tp1_msg_processed_request 867
can_tp1_msg_queue_level 855
can_tp1_msg_read 862
can_tp1_msg_rm_register 844
can_tp1_msg_rqrx_register 841
can_tp1_msg_rqtActivate 849
can_tp1_msg_rx_register 834
can_tp1_msg_send 852
can_tp1_msg_send_id_queued 857
can_tp1_msg_service_read 873
can_tp1_msg_service_request 872
can_tp1_msg_sleep 860
can_tp1_msg_trigger 864
can_tp1_msg_tx_register 830
can_tp1_msg_txqueue_init 855
can_tp1_msg_txqueue_level_read 859
can_tp1_msg_wakeup 861

can_tp1_msg_write 850
can_tp1_service_register 870
can_tp1_subint_handler_install 876
 common concepts 198
 Common Program Data folder 24
 compiler and C run-time libraries 926
 compiling an application 929

D

D/A conversion 259
 Analog Out Class 1 259
DacCl1AnalogOut_apply 260
DacCl1AnalogOut_create 261
DacCl1AnalogOut_setOutputValue 263
DacCl1AnalogOut_start 264
DacCl1AnalogOut_write 266
 data exchange functions 901
 data structures for CAN 807
 data type
 dsse_ISR 417
 dsse_LSR 419
 dsse_MSR 420
 dsse_subint_handler_t 421
 dsseChannel 422
DioCl1BcPwmOut_apply 727
DioCl1BcPwmOut_create 728
DioCl1BcPwmOut_disableInterrupts 731
DioCl1BcPwmOut_enableInterrupts 732
DioCl1BcPwmOut_setAcu 734
DioCl1BcPwmOut_setDirection 735
DioCl1BcPwmOut_setDutyCycle 737
DioCl1BcPwmOut_setEventDelay 738
DioCl1BcPwmOut_setEventDownsample 739
DioCl1BcPwmOut_setInterruptMode 741
DioCl1BcPwmOut_setIoIntVector 742
DioCl1BcPwmOut_setOutputHighZ 744
DioCl1BcPwmOut_setOutputMode 745
DioCl1BcPwmOut_setPeriod 747
DioCl1BcPwmOut_setPosition 748
DioCl1BcPwmOut_setPositionOffset 749
DioCl1BcPwmOut_setRisingEdgeDelay 751
DioCl1BcPwmOut_setSectorCount 752
DioCl1BcPwmOut_setSectorPositions 753
DioCl1BcPwmOut_setSectorSignals 754
DioCl1BcPwmOut_setSignalPairCheck 757
DioCl1BcPwmOut_setSignalVoltage 758
DioCl1BcPwmOut_setStationarySignal 759
DioCl1BcPwmOut_setTriggerLineOut 761
DioCl1BcPwmOut_setTriggerLineOutStatus 763
DioCl1BcPwmOut_setTriggerMode 764
DioCl1BcPwmOut_setUpdateMode 766
DioCl1BcPwmOut_start 767
DioCl1BcPwmOut_stop 768
DioCl1BcPwmOut_write 769
DioCl1DigIn_apply 269
DioCl1DigIn_create 270
DioCl1DigIn_disableInterrupts 272
DioCl1DigIn_enableInterrupts 273
DioCl1DigIn_getChMaskInData 274
DioCl1DigIn_getPortInData 275
DioCl1DigIn_read 276

DioCl1DigIn_setEventDelay 278
 DioCl1DigIn_setEventDownSample 279
 DioCl1DigIn_setInputFilter 280
 DioCl1DigIn_setInterruptMode 281
 DioCl1DigIn_setIoIntVector 282
 DioCl1DigIn_setTriggerLineOut 284
 DioCl1DigIn_setTriggerLineOutStatus 285
 DioCl1DigIn_setTriggerMode 286
 DioCl1DigIn_start 288
 DioCl1DigOut_apply 292
 DioCl1DigOut_create 293
 DioCl1DigOut_disableInterrupts 294
 DioCl1DigOut_enableInterrupts 295
 DioCl1DigOut_setChMaskOutData 297
 DioCl1DigOut_setChMaskOutHighZ 298
 DioCl1DigOut_setEventDelay 299
 DioCl1DigOut_setEventDownsample 300
 DioCl1DigOut_setInterruptMode 301
 DioCl1DigOut_setIoIntVector 303
 DioCl1DigOut_setPortOutData 304
 DioCl1DigOut_setPortOutHighZ 305
 DioCl1DigOut_setRisingEdgeDelay 307
 DioCl1DigOut_setSignalVoltage 308
 DioCl1DigOut_setTriggerLineOut 309
 DioCl1DigOut_setTriggerLineOutStatus 310
 DioCl1DigOut_setTriggerMode 311
 DioCl1DigOut_start 313
 DioCl1DigOut_write 314
 DioCl1EncoderIn_apply 606
 DioCl1EncoderIn_create 607
 DioCl1EncoderIn_getEncPosition 609
 DioCl1EncoderIn_getEncVelocity 611
 DioCl1EncoderIn_getIsIndexRaised 612
 DioCl1EncoderIn_getMechPosition 613
 DioCl1EncoderIn_getMechVelocity 614
 DioCl1EncoderIn_read 616
 DioCl1EncoderIn_setEncoderLines 617
 DioCl1EncoderIn_setEncPosition 618
 DioCl1EncoderIn_setEncPosValidity 619
 DioCl1EncoderIn_setGatedMode 621
 DioCl1EncoderIn_setIndexMode 622
 DioCl1EncoderIn_setIndexPosition 623
 DioCl1EncoderIn_setInputFilter 624
 DioCl1EncoderIn_setMaxPosition 626
 DioCl1EncoderIn_setMeasurementInterval 627
 DioCl1EncoderIn_setMinEncVelocity 628
 DioCl1EncoderIn_setMinMechVelocity 629
 DioCl1EncoderIn_setMinPosition 630
 DioCl1EncoderIn_start 632
 DioCl1EncoderIn_stop 633
 DioCl1EncoderIn_write 634
 DioCl1HallIn_apply 569
 DioCl1HallIn_create 570
 DioCl1HallIn_getIsPositionValid 573
 DioCl1HallIn_getPosition 574
 DioCl1HallIn_getSensorSignals 576
 DioCl1HallIn_read 577
 DioCl1HallIn_setInputFilter 578
 DioCl1HallIn_setPositionOffset 580
 DioCl1HallIn_setReverseDirection 581
 DioCl1HallIn_setSigEdgePositions 582
 DioCl1HallIn_start 583
 DioCl1HallIn_write 584
 DioCl1MultiPwmOut_apply 772
 DioCl1MultiPwmOut_create 774
 DioCl1MultiPwmOut_disableInterrupts 776
 DioCl1MultiPwmOut_enableInterrupts 777
 DioCl1MultiPwmOut_setAlignmentMode 779
 DioCl1MultiPwmOut_setDutyCycle 780
 DioCl1MultiPwmOut_setEventDelay 781
 DioCl1MultiPwmOut_setEventDownsample 783
 DioCl1MultiPwmOut_setInterruptMode 784
 DioCl1MultiPwmOut_setInvertingMode 785
 DioCl1MultiPwmOut_setIoIntVector 787
 DioCl1MultiPwmOut_setOutputHighZ 788
 DioCl1MultiPwmOut_setPeriod 790
 DioCl1MultiPwmOut_setRisingEdgeDelay 791
 DioCl1MultiPwmOut_setSignalVoltage 792
 DioCl1MultiPwmOut_setTriggerLineOut 794
 DioCl1MultiPwmOut_setTriggerLineOutStatus 795
 DioCl1MultiPwmOut_setTriggerMode 796
 DioCl1MultiPwmOut_setUpdateMode 798
 DioCl1MultiPwmOut_start 799
 DioCl1MultiPwmOut_write 801
 DioCl1PulseIn_apply 378
 DioCl1PulseIn_create 379
 DioCl1PulseIn_disableInterrupts 380
 DioCl1PulseIn_enableInterrupts 381
 DioCl1PulseIn_getPulseWidth 382
 DioCl1PulseIn_read 384
 DioCl1PulseIn_setEdgePolarity 385
 DioCl1PulseIn_setEventDelay 386
 DioCl1PulseIn_setInterruptMode 387
 DioCl1PulseIn_setIoIntVector 388
 DioCl1PulseIn_setTriggerLineOut 391
 DioCl1PulseIn_setTriggerLineOutStatus 392
 DioCl1PulseIn_setTriggerMode 390
 DioCl1PulseIn_setWidthMax 393
 DioCl1PulseIn_start 394
 DioCl1PulseOut_apply 365
 DioCl1PulseOut_create 366
 DioCl1PulseOut_setFirePulse 368
 DioCl1PulseOut_setInvertingMode 369
 DioCl1PulseOut_setOutputHighZ 370
 DioCl1PulseOut_setPulseWidth 371
 DioCl1PulseOut_setSignalVoltage 372
 DioCl1PulseOut_setTriggerLineIn 373
 DioCl1PulseOut_start 374
 DioCl1PulseOut_write 375
 DioCl1PwmIn_apply 344
 DioCl1PwmIn_create 345
 DioCl1PwmIn_disableInterrupts 346
 DioCl1PwmIn_enableInterrupts 347
 DioCl1PwmIn_getDutyCycle 349
 DioCl1PwmIn_getFrequency 350
 DioCl1PwmIn_read 351
 DioCl1PwmIn_setEventDelay 352
 DioCl1PwmIn_setEventDownsample 353
 DioCl1PwmIn_setInterruptMode 354
 DioCl1PwmIn_setIoIntVector 356
 DioCl1PwmIn_setTimeout 357
 DioCl1PwmIn_setTriggerLineOut 360
 DioCl1PwmIn_setTriggerLineOutStatus 361
 DioCl1PwmIn_setTriggerMode 358
 DioCl1PwmIn_setUpdateMode 362
 DioCl1PwmIn_start 363
 DioCl1PwmOut_apply 319
 DioCl1PwmOut_create 320
 DioCl1PwmOut_disableInterrupts 321
 DioCl1PwmOut_enableInterrupts 322
 DioCl1PwmOut_setDutyCycle 324
 DioCl1PwmOut_setEventDelay 325
 DioCl1PwmOut_setEventDownsample 326
 DioCl1PwmOut_setInterruptMode 327
 DioCl1PwmOut_setInvertingMode 328
 DioCl1PwmOut_setIoIntVector 329
 DioCl1PwmOut_setOutputHighZ 330
 DioCl1PwmOut_setPeriod 332
 DioCl1PwmOut_setRisingEdgeDelay 333
 DioCl1PwmOut_setSignalVoltage 334
 DioCl1PwmOut_setTriggerLineOut 335
 DioCl1PwmOut_setTriggerLineOutStatus 336
 DioCl1PwmOut_setTriggerMode 338
 DioCl1PwmOut_setUpdateMode 339
 DioCl1PwmOut_start 340
 DioCl1PwmOut_write 341
 DioCl1Spi_apply 453
 DioCl1Spi_create 454
 DioCl1Spi_disableInterrupts 456
 DioCl1Spi_enableInterrupts 457
 DioCl1Spi_setChipSelectPolarity 459
 DioCl1Spi_setInterruptMode 460
 DioCl1Spi_setIoIntVector 461
 DioCl1Spi_setOutputsHighZ 462
 DioCl1Spi_setSignalVoltage 463
 DioCl1Spi_setTriggerLineOut 466
 DioCl1Spi_setTriggerLineOutStatus 467
 DioCl1Spi_setTriggerMode 464
 DioCl1Spi_start 468
 DioCl1Spi_write 469
 DioCl1SpiCycle_apply 470
 DioCl1SpiCycle_create 471
 DioCl1SpiCycle_getCycleData 473
 DioCl1SpiCycle_read 475
 DioCl1SpiCycle_setBaudRate 476
 DioCl1SpiCycle_setBitDirection 477
 DioCl1SpiCycle_setChipSelectConf 478
 DioCl1SpiCycle_setClockPhase 480
 DioCl1SpiCycle_setClockPolarity 479
 DioCl1SpiCycle_setCycleData 482
 DioCl1SpiCycle_setTimeAfterTran 483
 DioCl1SpiCycle_setTimeBeforeTran 484
 DioCl1SpiCycle_setTimeBetwWords 486
 DioCl1SpiCycle_setTimeCslnactive 487
 DioCl1SpiCycle_start 488
 DioCl1SpiCycle_write 489
 DioCl2EncoderIn_apply 637
 DioCl2EncoderIn_create 638
 DioCl2EncoderIn_getEncPosition 640
 DioCl2EncoderIn_getEncVelocity 642
 DioCl2EncoderIn_getIsIndexRaised 643
 DioCl2EncoderIn_getMechPosition 644

DioCl2EncoderIn_getMechVelocity 645
 DioCl2EncoderIn_read 647
 DioCl2EncoderIn_setEncoderLines 648
 DioCl2EncoderIn_setEncPosition 649
 DioCl2EncoderIn_setEncPosValidity 650
 DioCl2EncoderIn_setGatedMode 652
 DioCl2EncoderIn_setIndexMode 653
 DioCl2EncoderIn_setIndexPosition 654
 DioCl2EncoderIn_setInputFilter 655
 DioCl2EncoderIn_setMaxPosition 657
 DioCl2EncoderIn_setMeasurementInterval 658
 DioCl2EncoderIn_setMinEncVelocity 659
 DioCl2EncoderIn_setMinMechVelocity 660
 DioCl2EncoderIn_setMinPosition 661
 DioCl2EncoderIn_start 663
 DioCl2EncoderIn_stop 664
 DioCl2EncoderIn_write 665
 DioCl2EnDatIn_apply 537
 DioCl2EnDatIn_create 538
 DioCl2EnDatIn_getConfigErr 540
 DioCl2EnDatIn_getCrcErrorCount 542
 DioCl2EnDatIn_getIsPositionValid 543
 DioCl2EnDatIn_getIsRevoltNoValid 544
 DioCl2EnDatIn_getMechPosition 546
 DioCl2EnDatIn_getRevolutionNo 548
 DioCl2EnDatIn_getTransmissionErr 549
 DioCl2EnDatIn_read 551
 DioCl2EnDatIn_setClockFrequency 552
 DioCl2EnDatIn_setCrcErrorLimit 555
 DioCl2EnDatIn_setMultiturnRes 556
 DioCl2EnDatIn_setNumberOfSteps 558
 DioCl2EnDatIn_setPositionOffset 559
 DioCl2EnDatIn_setReverseDirection 560
 DioCl2EnDatIn_setSampleSyncEvSrc 562
 DioCl2EnDatIn_setSingleturnRes 563
 DioCl2EnDatIn_start 565
 DioCl2EnDatIn_stop 566
 DioCl2HallIn_apply 587
 DioCl2HallIn_create 588
 DioCl2HallIn_getIsPositionValid 590
 DioCl2HallIn_getPosition 592
 DioCl2HallIn_getSensorSignals 593
 DioCl2HallIn_read 594
 DioCl2HallIn_setInputFilter 596
 DioCl2HallIn_setPositionOffset 597
 DioCl2HallIn_setReverseDirection 598
 DioCl2HallIn_setSigEdgePositions 599
 DioCl2HallIn_start 601
 DioCl2HallIn_write 602
 DioCl2Ssiln_apply 687
 DioCl2Ssiln_create 689
 DioCl2Ssiln_getCommunicationErr 691
 DioCl2Ssiln_getIsPositionValid 692
 DioCl2Ssiln_getIsRevoltNoValid 694
 DioCl2Ssiln_getMechPosition 695
 DioCl2Ssiln_getRevolutionNo 697
 DioCl2Ssiln_getSsiFrameData 698
 DioCl2Ssiln_getTransmitErrCount 700
 DioCl2Ssiln_read 701
 DioCl2Ssiln_setClockFrequency 703
 DioCl2Ssiln_setClockPause 705

DioCl2Ssiln_setCodeType 706
 DioCl2Ssiln_setMultiturnRes 707
 DioCl2Ssiln_setNoOfFirstPosBit 709
 DioCl2Ssiln_setNumberOfSteps 710
 DioCl2Ssiln_setPositionOffset 712
 DioCl2Ssiln_setRepeatReading 713
 DioCl2Ssiln_setReverseDirection 715
 DioCl2Ssiln_setSampleSyncEvSrc 716
 DioCl2Ssiln_setSingleturnRes 718
 DioCl2Ssiln_setTransmitBitCount 719
 DioCl2Ssiln_setTransmitErrLimit 721
 DioCl2Ssiln_start 722
 DioCl2Ssiln_stop 723
 Documents folder 24
 down1202 930
 downloading an application 929
 DsBuildApplication.mk 933
 DsBuildLoad.mk 934
 DsBuildTemplate.mk 934
 dsse_bytes2word 448
 dsse_config 427
 dsse_disable 436
 dsse_enable 436
 dsse_error_read 437
 dsse_fifo_reset 435
 dsse_free 426
 dsse_handle_get 441
 dsse_init 425
 dsse_ISR 417
 dsse_LSR 419
 dsse_MSR 420
 dsse_receive 432
 dsse_receive_fifo_level 439
 dsse_receive_term 433
 dsse_set 442
 dsse_status_read 440
 dsse_subint_disable 445
 dsse_subint_enable 444
 dsse_subint_handler_inst 443
 dsse_subint_handler_t 421
 dsse_transmit 430
 dsse_transmit_fifo_level 438
 dsse_word2bytes 446
 dsseChannel 422
 dssint_acknowledge 127
 dssint_decode 126
 dssint_define_int_receiver 120
 dssint_define_int_receiver_1 122
 dssint_define_int_sender 116
 dssint_define_int_sender_1 118
 dssint_interrupt 126
 dssint_subint_disable 124
 dssint_subint_enable 125
 dssint_subint_reset 128

E

electric motor control functions 491
 elementary data types 26
 EnDat interface
 class 2 536
 environment variables and paths 926

error messages 131
 example
 using time measurement functions 34
 examples
 using CAN 880

F

folder structure 927
 FPGA initialization 890
 FPGA Type 1
 MicroLabBox 889
 RTLib functions 889

H

Hall sensor 568
 class 1 568
 class 2 586
 host programs 925
 host settings 926

I

I/O functions 197
 identifying IOFPGA components 892
 incremental encoder 604
 class 1 604
 class 2 635
 information messages 131
 init
 MicroLabBox 28
 init() 187
 interrupt
 flag 99
 handling 93
 receiver 110
 sender 110
 IOFPGA interrupt functions 896
 IoFpga_ack_int 898
 IoFpga_disable_int 897
 IoFpga_enable_int 896
 IoFpga_get_appl_id 894
 IoFpga_get_appl_id_string 895
 IoFpga_get_board_rev 892
 IoFpga_get_fw_id 893
 IoFpga_init 890
 IoFpga_read_buf 917
 IoFpga_read_buf64 919
 IoFpga_read_reg 902
 IoFpga_read_reg_grp 908
 IoFpga_read_reg_grp_mixed 910
 IoFpga_read_reg64 903
 IoFpga_register_isr 899
 IoFpga_write_buf 921
 IoFpga_write_buf64 923
 IoFpga_write_reg 905
 IoFpga_write_reg_grp 912
 IoFpga_write_reg_grp_mixed 914
 IoFpga_write_reg64 906

L

LED control 162
 Led_apply 163
 Led_create 164
 Led_setBlueColor 165
 Led_setGreenColor 167
 Led_setRedColor 168
 Led_start 169
 Led_write 170
 linking an application 929
 Local Program Data folder 24

M

macro definition 186
 message
 module 130
 message buffer 131
 message handling 130, 131
 message length 131
 message type 131
 MicroLabBox
 init 28
 msg_default_dialog_set 144
 msg_error_clear 149
 msg_error_hook_set 150
 msg_error_printf 138
 msg_error_set 134
 msg_info_printf 141
 msg_info_set 135
 msg_init 151
 msg_last_error_number 146
 msg_last_error_submodule 147
 msg_mode_set 145
 msg_printf 142
 msg_reset 146
 msg_set 136
 msg_warning_printf 140
 msg_warning_set 135
 multichannel PWM signal generation
 class 1 771

N

nonvolatile data handling 399
 ntoppc-objdump 937
 NvData
 Example 409
 RTLib functions 399
 NvData_apply 400
 NvData_create 401
 NvData_createDataSet 402
 NvData_read 403
 NvData_setDimension 404
 NvData_setName 405
 NvData_setType 407
 NvData_write 408

R

receive buffer 412
 receiver type definition 115

Resolver interface 667
 ResolverIn_apply 668
 ResolverIn_create 669
 ResolverIn_getFaultStatus 671
 ResolverIn_getIsDataValid 673
 ResolverIn_getPosition 674
 ResolverIn_setExcitationFreq 677
 ResolverIn_setExcitationVoltRms 678
 ResolverIn_setMaximumSpeed 679
 ResolverIn_setPositionOffset 680
 ResolverIn_setReverseDirection 682
 ResolverIn_setSineCosineVoltRms 683
 ResolverIn_start 684
 rtplib_background_hook 29
 RTLIB_BACKGROUND_SERVICE 29
 RTLIB_CALLOC_PROT 193
 RTLIB_CONV_FLOAT_TO_SATURATED_INT32 184
 RTLIB_CONV_FLOAT32_FROM_IEEE32 182
 RTLIB_CONV_FLOAT32_FROM_TI32 183
 RTLIB_CONV_FLOAT32_TO_IEEE32 183
 RTLIB_CONV_FLOAT32_TO_TI32 184
 RTLIB_FORCE_IN_ORDER 180
 RTLIB_FREE_PROT 194
 RTLIB_GET_SERIAL_NUMBER 189
 RTLIB_INIT 188
 RTLIB_INT_DISABLE 106
 RTLIB_INT_ENABLE 106
 RTLIB_INT_RESTORE 107
 RTLIB_INT_SAVE_AND_DISABLE 107
 RTLIB_MALLOC_PROT 192
 RTLIB_REALLOC_PROT 194
 RTLIB_REGISTER_BACKGROUND_HANDLER 190
 RTLIB_REGISTER_TERMINATE_HANDLER 191
 RTLIB_REGISTER_UNLOAD_HANDLER 191
 RTLIB_SRT_DISABLE 108
 RTLIB_SRT_ENABLE 109
 RTLIB_SRT_ISR_BEGIN 90
 RTLIB_SRT_ISR_END 91
 RTLIB_SRT_PERIOD 66
 RTLIB_SRT_START 91
 RTLIB_SYNC 181
 RTLIB_TERMINATE 189
 RTLIB_TIC_CONTINUE 44
 RTLIB_TIC_COUNT 44
 RTLIB_TIC_DELAY 45
 RTLIB_TIC_DIFF 46
 RTLIB_TIC_ELAPSED 47
 RTLIB_TIC_HALT 48
 RTLIB_TIC_READ 49
 RTLIB_TIC_READ_TOTAL 50
 RTLIB_TIC_START 50

S

sender type definition 115
 sensor supply 171
 SensorSupply_apply 171
 SensorSupply_create 173
 SensorSupply_setSensorState 174
 SensorSupply_setVoltage 175
 SensorSupply_start 177

SensorSupply_write 178
 serial interface communication 411
 Serial Peripheral Interface 451
 slave CAN
 communication channel 804
 slave CAN access functions 804
 srtk_begin_isr_timerA 83
 srtk_begin_isr_timerB 84
 srtk_begin_isr_timerD 85
 srtk_disable_hardware_int 94
 srtk_disable_hardware_int_bm 95
 srtk_enable_hardware_int 96
 srtk_enable_hardware_int_bm 97
 srtk_end_isr_timerA 85
 srtk_end_isr_timerB 86
 srtk_end_isr_timerD 86
 srtk_get_interrupt_flag 99
 srtk_get_interrupt_flag_bm 100
 srtk_get_interrupt_vector 101
 srtk_reset_interrupt_flag 102
 srtk_reset_interrupt_flag_bm 103
 srtk_set_interrupt_vector 104
 srtk_start_isr_timerA 87
 srtk_start_isr_timerB 88
 srtk_start_isr_timerD 89
 srtk_tic_continue 34
 srtk_tic_count 35
 srtk_tic_delay 36
 srtk_tic_diff 37
 srtk_tic_elapsed 38
 srtk_tic_halt 39
 srtk_tic_read 40
 srtk_tic_start 41
 srtk_tic_total_read 41
 srtk_timebase_fltread 42
 srtk_timebase_low_read 42
 srtk_timebase_read 43
 srtk_timerA_period_reload 67
 srtk_timerA_period_set 67
 srtk_timerA_read 68
 srtk_timerA_start 69
 srtk_timerA_stop 69
 srtk_timerB_compare_set 73
 srtk_timerB_compare_set_periodically 74
 srtk_timerB_init 72
 srtk_timerB_read 75
 srtk_timerB_start 75
 srtk_timerB_stop 76
 srtk_timerD_period_reload 79
 srtk_timerD_period_set 78
 srtk_timerD_read 79
 srtk_timerD_start 80
 srtk_timerD_stop 81
 SSI interface
 class 2 686
 subinterrupt
 serial communication 413
 subinterrupt handling 110
 System functions 153

T

Time Base Counter 32
time interval measurement 32
time measurement example 34
time stamping module 52
timebase counter 32
Timer A 65
 example 65
Timer B 71
 example 72
Timer D 77
timer interrupt control 82
Timing I/O 317, 342, 364, 376
transmit buffer 412
trigger level 413
ts_init 55
ts_mat_period_get 57
ts_mit_period_get 57
ts_reset 58
ts_time_calculate 63
ts_time_offset 61
ts_time_read 58
ts_timestamp_calculate 64
ts_timestamp_compare 60
ts_timestamp_interval 61
ts_timestamp_offset 62
ts_timestamp_read 59

U

UART 411
USB Flight Recorder 397
using CAN
 examples 880

W

warning messages 131

