

Diagnostic state machine error codes

Details about the errors of the diagnostic state machine.

They are either returned by the **DIAG_Status(..)** function or by the callback function (**DIAG_ERR_CALLBACK**).

The following table gives an overview of the error codes and their types.

There are **four types of errors**:

- **Fatal errors** If a fatal error is detected, the diagnostic state machine always activates the safe state.
- **Permanent errors**
For non-fatal errors the application can decide which action to take (via the error callback, see **DIAG_ERR_CALLBACK**).
- **Temporary fatal errors**
For these errors a glitch filter (de-bounce) is implemented, which means that they are reported after the configured anti-glitch time. After the glitch filter time has expired, the error will be treated like a fatal error.
- **temporary permanent errors**
For these errors a glitch filter (de-bounce) is implemented, which means that they are reported after the configured anti-glitch time. After the glitch filter time has expired, the error will be treated like a permanent error.

For the error values please refer to **Diagnostic State Machine Error Values**

Name	fatal?	temporary?	short description	content of value
DIAG_E_NOERROR	TRUE	FALSE	no error	not available
DIAG_E_ADC_LIMITS	FALSE	TRUE	an ADC limit was exceeded or underrun	Measured voltage value in [mV]
DIAG_E_ADC_5V2_SUPPLY	TRUE	TRUE	the 5.2V supply voltage deviates from its normal level 4900mV - 5380mV	Measured supply voltage in [mV]
DIAG_E_ADC_SENSOR_SUPPLY	FALSE	TRUE	the sensor supply voltage deviates from its normal level 4850mV - 5150mV	Measured sensor supply voltage in [mV]
DIAG_E_ADC_KL30_MAIN	FALSE	TRUE	the KL30 voltage level exceeds or underruns its	Measured battery voltage in [mV]

			allowed limits 7600mV - 33400mV	
DIAG_E_ADC_KL30_CPU	TRUE	TRUE	the KL30_CPU voltage level exceeds or underruns its allowed limits 7600mV - 33400mV	Measured b: voltage for tl [mV]
DIAG_E_OVER_TEMPERATURE	TRUE	TRUE	the board temperature exceeds its allowed range (the board temperature is 133 degree Celcius or higher)	Board tempe tenth degree <i>Remark:</i> Fai is signed (tw complement
DIAG_E_MEM_USER_STACK	TRUE	FALSE	a memory block of the user stack is corrupted	The faulty va is written to i protected sp stack memo
DIAG_E_MEM_REGISTER	TRUE	FALSE	internal register values are corrupted	not available
DIAG_E_MEM_DSRAM	TRUE	FALSE	a memory block of the data S-RAM is corrupted	The divergin that has bee detected
DIAG_E_MEM_PSRAM	TRUE	FALSE	a memory block of the program S- RAM is corrupted	The divergin that has bee detected
DIAG_E_MEM_DPRAM	TRUE	FALSE	a memory block of the dual-port RAM is corrupted	The divergin that has bee detected
DIAG_E_MEM_ZeroFlag	TRUE	FALSE	the zero flag was mistakenly raised during a math operation	Register PS' XC2000 CPI
DIAG_E_MEM_CarryFlag	TRUE	FALSE	the carry flag was mistakenly raised during a math operation	Register PS' XC2000 CPI
DIAG_E_MEM_NegativeFlag	TRUE	FALSE	the negative flag was mistakenly raised during a math operation	Register PS' XC2000 CPI
DIAG_E_MEM_OverflowFlag	TRUE	FALSE	the overflow flag was mistakenly raised during a math operation	Register PS' XC2000 CPI
DIAG_E_MEM_SYS_STACK_OF	TRUE	FALSE	An overflow of the system stack	System stac of the XC201

			occurred	
DIAG_E_MEM_SYS_STACK_UF	TRUE	FALSE	An underflow of the system stack occurred	System stack of the XC2000
DIAG_E_MEM_SR0_TRAP	TRUE	FALSE	A SR0 trap occurred and all interrupts have been disabled. For further information about System-Request-0 traps consult the XC2000 user manual.	Register SCU.TRAPS the XC2000
DIAG_E_MEM_CLASS_B_TRAP	TRUE	FALSE	<p>A class B trap occurred and all interrupts have been disabled.</p> <p><i>Examples for class B traps are:</i></p> <ul style="list-style-type: none"> - Undefined Opcode - Memory Access Error - Illegal Word Operand Access <p>For further information about Class B traps consult the XC2000 user manual.</p>	Register TFI XC2000 CPI
DIAG_E_FREQ_STARTUP	TRUE	FALSE	Internal clock frequency drifts from its normal value	Indication if frequency is or too low
DIAG_E_PWM_CURRENT_ZERO	FALSE	FALSE	Current measurement greater zero during startup phase	Measured current [digits * number_of_digits] or 0xFFFF if a synchronization error between sensor and checker has been detected
DIAG_E_PWM_CURRENT_OFFSET	FALSE	FALSE	Offset of current measurement circuitry on PWM output not within limits	Offset of the measurement stored in the [digits * number_of_digits] This value is scaled to 1 number of digits

				taken for one mea
DIAG_E_PWM_LIMITS_RANGE	FALSE	TRUE	Pulse width not within range on PWM output (outside min/max pulse)	Read duty c
DIAG_E_PWM_LIMITS_TOL	FALSE	TRUE	Pulse width not within tolerance window on PWM output	Difference b set and read cycle in [us]
DIAG_E_PWM_PERIOD_MISMATCH	FALSE	TRUE	Period mismatch on PWM output	Difference b set and read [us]
DIAG_E_PWM_CURRENT	FALSE	TRUE	Current not within limits on current controlled input	Difference b set and read [mA]
DIAG_E_PWM_CURRENT_DEAD_TIME	FALSE	FALSE	Set current not reached after dead time elapsed	Difference b set and read [mA]
DIAG_E_PWM_CURRENT_OFFSETS_DRIFT	FALSE	TRUE	Current offset too low (due to drift or HW defect)	Measured ra in [digits * number_of_
DIAG_E_PWD_LIMITS_FREQ	FALSE	TRUE	Frequency limit error on timer input or counter limit error in case of incremental/counter input	Measured fr value in [Hz] counter valu [digits]
DIAG_E_PWD_LIMITS_PULSE_WIDTH	FALSE	TRUE	Pulse width limit error on timer input	Measured p in [us]. This value is satu 0xFFFF
DIAG_E_CYCLE_TIME	FALSE	FALSE	Cycle time too high	Measured cy in [us]. This value is satu 0xFFFF
DIAG_E_RPP	FALSE	TRUE	Insufficient gate drive on reverse polarity protection. <i>Further Reasons for occurrence:</i> - Clamp 30 voltage is not connected to ECU - Driver is used with external safety switch but no PWM is configured safety critical	Difference b Urpp and Ut (Urpp .. Gate reverse pola protection, Ubat .. Supp for output st

DIAG_E_EXT_WD	TRUE	FALSE	External WD has activated the safe state	Content of fl in register SCU.DMPM XC2000 CPI
DIAG_E_LS_PROT	FALSE	TRUE	Over-current condition on safety switch	Error code (IO_ErrorTy digital output)
DIAG_E_OVD_STARTUP	TRUE	FALSE	Over voltage detector startup test has failed <i>Further Reasons for occurrence:</i> - Hardware over-current protection is tripped due to a short battery condition on the low-side digital outputs	Internal state information (which an error occurred)
DIAG_E_OVD	TRUE	FALSE	Over voltage detection has activated the safe state	Read status from OVD ci
DIAG_E_SAFETY_SW_INT	TRUE	FALSE	Safety switch check error (internal switch) <i>Further Reasons for occurrence:</i> - Improper wiring - Improper safety switch setting for safety-critical PWM output - Driver has been configured for use with an external safety switch but an internal one or no safety switch is wired	Internal state information (which an error occurred)
DIAG_E_SAFETY_SW_EXT	TRUE	FALSE	Safety switch check error (external switch) <i>Further Reasons for occurrence:</i> - Improper wiring of external safety switch - Improper safety switch setting for safety-critical PWM output	Internal state information (which an error occurred)

			<p>- Driver has been configured for internal safety switch but an external one is wired</p> <p>Attention</p> <p>The startup test for the external Safety Switch runs in a blocking manner when being already in the cyclic execution phase (when the application software is already running). The startup test will violate the cycle time that is defined by the application software.</p>	
DIAG_E_SAFETY_SW_SHUT_OFF	FALSE	FALSE	An error while checking the PWM feedback shut off path of the safety switches occurred	Internal state information (which an error occurred)
DIAG_E_INIT_ERROR	TRUE	FALSE	Error during initialization of diagnostic state machine	IO-Driver error
DIAG_E_INVALID_DIAG_STATE	TRUE	FALSE	Invalid diagnostic state	Content of state variable which the diagnostic state machine is in
DIAG_E_INVALID_STARTUP_STATE	TRUE	FALSE	Invalid diagnostic state in configuration state	Content of state variable which the diagnostic state machine is in
DIAG_E_INVALID_MAIN_STATE	TRUE	FALSE	Invalid diagnostic state in main state	Content of state variable which the diagnostic state machine is in
DIAG_E_WD_STARTUP	TRUE	FALSE	Watchdog startup test has failed	

DIAG_E_SR_LowNibble	TRUE	TRUE	The feedback value of the low nibble of the shift register is faulty	
DIAG_E_SR_HighNibble	TRUE	TRUE	The feedback value of the high nibble of the shift register is faulty	
DIAG_E_TIMEOUT	TRUE	FALSE	A timeout between Diagnostic Module and IO-Driver has occurred (e.g. driver functions of safety critical IO not called)	not available
DIAG_E_APPL_SAFE_STATE	TRUE	FALSE	Application requested to activate the safe state	not available
DIAG_E_PLL_VCO_NOT_LOCKED	TRUE	FALSE	The PLL/VCO lost its lock to the oscillator frequency. (e.g. if the oscillator is damaged or not connected to the CPU anymore)	Content of C register PLL
DIAG_E_SW_INTERNAL	TRUE	FALSE	Internal SW error detected. May be SW or HW related.	not available
DIAG_E_INIT_ERROR	TRUE	FALSE	Error during initialization of diagnostic state machine	Error code IO_E_SW_IL
DIAG_E_INT_WATCHDOG	TRUE	FALSE	Internal Watchdog has not been serviced in time - WD reset occurred.	Content of C register RST



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

[Main Page](#)[Related Pages](#)[Data Structures](#)[Files](#)

HY-TTC30 Family pin features

Listing of all IO driver pins and their configuration options

Analog Inputs

1-Mode ADC inputs (primary function)

These pins can be used as ADC inputs:

- `IO_ADC_20`
- `IO_ADC_21`

They can be configured by software for:

- *absolute* (0 .. 32V)

For details on the usage refer to the **driver functions for analog inputs**.

As a secondary function they can be used as:

- **Digital Inputs**

3-Mode ADC inputs (primary function)

These pins can be used as ADC inputs:

- `IO_ADC_10`
- `IO_ADC_11`
- `IO_ADC_12`
- `IO_ADC_13`
- `IO_ADC_14` (for HY-TTC32 variants pin supports also resistive mode)
- `IO_ADC_15` (for HY-TTC32 variants pin supports also resistive mode)

They can be configured by software for:

- *absolute* (0 .. 5V or 0 .. 10V)
- *ratiometric* (0 .. 5V)
- *current* (0 .. 24mA)

For details on the usage refer to the **driver functions for analog inputs**.

As a secondary function they can be used as:

- **Digital Inputs**

4-Mode ADC inputs (primary function)

These pins can be used as ADC inputs:

- `IO_ADC_00`
- `IO_ADC_01`

They can be configured by software for:

- *absolute* (0 .. 5V or 0 .. 10V)
- *ratiometric* (0 .. 5V)
- *current* (0 .. 24mA)
- *resistive* (0 .. 65kOhm)

For details on the usage refer to the **driver functions for analog inputs**.

As a secondary function they can be used as:

- **Digital Inputs**

Normal ADC inputs

These pins have a fixed assignment to internal board voltages:

- `IO_ADC_UBAT`
- `IO_ADC_UBAT_CPU`
- `IO_ADC_SENSOR_SUPPLY`
- `IO_K15`
- `IO_ADC_NODE_ID_0`
- `IO_ADC_NODE_ID_1`
- `IO_ADC_BOARD_TEMP`

For details on the usage refer to the **driver functions for analog inputs**.

32V Analog Inputs (secondary function of PWM Outputs)

As secondary function, these pins can be used as Analog Inputs:

- `IO_ADC_34`
- `IO_ADC_35`
- `IO_ADC_36`
- `IO_ADC_37`
- `IO_ADC_38`
- `IO_ADC_39`
- `IO_ADC_40`
- `IO_ADC_41`

They can be configured by software for:

- *absolute (0 .. 32V)*

LED Outputs

One of the following:

- `IO_LED_00`
- `IO_LED_01`
- `IO_LED_02`
- `IO_LED_03`
- `IO_LED_04`
- `IO_LED_05`
- `IO_LED_06`
- `IO_LED_07`

32V Analog Inputs (secondary function of Low-Side Digital Outputs)

As secondary function, these pins can be used as Analog Inputs:

- `IO_ADC_28`
- `IO_ADC_29`

They can be configured by software for:

- *absolute* (0 .. 32V)

32V Analog Inputs (secondary function of PWD Inputs)

As secondary function, these pins can be used as Analog Inputs:

- `IO_ADC_30`
- `IO_ADC_31`
- `IO_ADC_32`
- `IO_ADC_33`

They can be configured by software for:

- *absolute (0 .. 32V)*

32V Analog Inputs (secondary function of PVG Outputs)

As secondary function, these pins can be used as Analog Inputs:

- `IO_ADC_22`
- `IO_ADC_23`
- `IO_ADC_24`
- `IO_ADC_25`
- `IO_ADC_26`
- `IO_ADC_27`

They can be configured by software for:

- *absolute (0 .. 32V)*

Digital inputs

Digital inputs with configurable pull-up/down resistor (secondary function of 1-Mode ADC Inputs)

As secondary function, these pins can be used as digital inputs with a voltage range of 0-32V and have a configurable pull-up/down resistor:

- `IO_DI_02`
- `IO_DI_03`

For details on the usage refer to the **driver functions for digital inputs and outputs**.

Digital inputs (secondary function of 3-Mode ADC Inputs)

As secondary function, these pins can be used as digital inputs with a voltage range of 0-10V:

- `IO_DI_10`
- `IO_DI_11`
- `IO_DI_12`
- `IO_DI_13`
- `IO_DI_14`
- `IO_DI_15`

For details on the usage refer to the **driver functions for digital inputs and outputs**.

Digital inputs (secondary function of 4-Mode ADC Inputs)

As secondary function, these pins can be used as digital inputs with a voltage range of 0-10V:

- `IO_DI_00`
- `IO_DI_01`

For details on the usage refer to the **driver functions for digital inputs and outputs**.

Digital inputs with configurable pull-up/down resistor (secondary function of PWD Inputs)

As secondary function, these pins can be used as digital inputs with a voltage range of 0-32V and have a configurable pull-up/down resistor:

- `IO_DI_04`
- `IO_DI_05`
- `IO_DI_06`
- `IO_DI_07`

For details on the usage refer to the **driver functions for digital inputs and outputs**.

Digital inputs (secondary function of PWM Outputs)

As secondary function, these pins can be used as digital inputs with a voltage range of 0-32V:

- `IO_DI_24`
- `IO_DI_25`
- `IO_DI_26`
- `IO_DI_27`
- `IO_DI_28`
- `IO_DI_29`
- `IO_DI_30`
- `IO_DI_31`

For details on the usage refer to the **driver functions for digital inputs and outputs**.

Digital inputs (secondary function of Low-Side Digital Outputs)

As secondary function, these pins can be used as digital inputs with a voltage range of 0-32V:

- `IO_DI_22`
- `IO_DI_23`

For details on the usage refer to the **driver functions for digital inputs and outputs**.

Digital inputs (secondary function of PVG Outputs)

As secondary function, these pins can be used as digital inputs with a voltage range of 0-32V:

- `IO_DI_16`
- `IO_DI_17`
- `IO_DI_18`
- `IO_DI_19`
- `IO_DI_20`
- `IO_DI_21`

For details on the usage refer to the **driver functions for digital inputs and outputs**.

PWD Inputs

Complex Digital Timer Inputs with configurable pull-up/down resistor and incremental decoder (primary function)

These pins can be used as digital timer inputs with incremental decoding:

- `IO_PWD_00`
- `IO_PWD_01`

They can decode

- *signals from incremental encoders,*
- *PWM signals or*
- *signals from ABS sensors.*

Furthermore they provide analog (ADC) feedback. As PWM decoder the frequency and pulse width can be measured at the same time.

As a secondary function they can be used as:

- **Analog Inputs**
- **Digital Inputs**

Complex Digital Timer Inputs with configurable pull-up/down resistor (primary function)

These pins can be used as digital timer inputs:

- `IO_PWD_02`
- `IO_PWD_03`

They can decode

- *PWM signals* or
- *signals from ABS sensors.*

Furthermore they provide analog (ADC) feedback. As PWM decoder the frequency and pulse width can be measured at the same time.

As a secondary function they can be used as:

- **Analog Inputs**
- **Digital Inputs**

Complex Digital Timer Inputs (secondary function of PWM Outputs)

As secondary function these pins can be used as digital timer inputs:

- `IO_PWD_22`
- `IO_PWD_23`

They can decode

- *PWM signals* or
- *signals from ABS sensors.*

Furthermore they provide analog (ADC) feedback. As PWM decoder the frequency and pulse width can be measured at the same time.

For details on the usage refer to the **driver functions for digital timer inputs**.

Digital Timer Inputs (secondary function of PWM Outputs with Current Measurement)

As secondary function these pins can be used as digital timer inputs to decode PWM signals. They can measure either the frequency or the pulse duration:

- `IO_PWD_10`
- `IO_PWD_11`
- `IO_PWD_12`
- `IO_PWD_13`
- `IO_PWD_20`
- `IO_PWD_21`

Furthermore they provide analog (ADC) feedback.

For details on the usage refer to the **driver functions for digital timer inputs**.

High-Side PWM outputs

High-Side PWM Outputs with Current Measurement (primary function)

These pins can be used to generate a pulse width modulated (PWM) output:

- `IO_PWM_00`
- `IO_PWM_01`
- `IO_PWM_02`
- `IO_PWM_03`
- `IO_PWM_04`
- `IO_PWM_05`

Furthermore they provide current measurement (CM), digital timer (PWD) and analog (ADC) feedback.

For details on the usage refer to the **driver functions for PWM outputs**.

As a secondary function they can be used as:

- **Analog Inputs**
- **Digital Inputs**
- **PWD Inputs**
- **Digital Outputs**

High-Side PWM Outputs (primary function)

These pins can be used to generate a pulse width modulated (PWM) output:

- `IO_PWM_10`
- `IO_PWM_11`

Furthermore they provide a over-current monitoring to detect overload situations, digital timer (PWD) and analog (ADC) feedback.

For details on the usage refer to the **driver functions for PWM outputs**.

As a secondary function they can be used as:

- **Analog Inputs**
- **Digital Inputs**
- **PWD Inputs**
- **Digital Outputs**

Digital Outputs

Low-Side Digital Outputs (primary function)

These pins can be used as digital low-side outputs:

- `IO_DO_10`
- `IO_DO_11`

For details on the usage refer to the **driver functions for digital inputs and outputs**.

As a secondary function they can be used as:

- **Analog Inputs**
- **Digital Inputs**

High-Side Digital Outputs with current measurement (secondary function of PWM Outputs)

These pins can be used as digital high-side switches:

- `IO_DO_20`
- `IO_DO_21`
- `IO_DO_22`
- `IO_DO_23`
- `IO_DO_24`
- `IO_DO_25`

Furthermore they provide current measurement (CM) and analog (ADC) feedback.

For details on the usage refer to the **driver functions for digital inputs and outputs**.

High-Side Digital Outputs (secondary function of PWM Outputs)

These pins can be used as digital high-side switches:

- `IO_DO_00`
- `IO_DO_01`
- `IO_DO_02`
- `IO_DO_03`
- `IO_DO_04`
- `IO_DO_05`
- `IO_DO_06`
- `IO_DO_07`

Furthermore they provide an over-current monitoring to detect overload situations and analog (ADC) feedback.

For details on the usage refer to the **driver functions for digital inputs and outputs**.

Push-Pull Digital Outputs (secondary function of PVG Outputs)

These pins can be used as push-pull digital output switches:

- `IO_DO_30`
- `IO_DO_31`
- `IO_DO_32`
- `IO_DO_33`
- `IO_DO_34`
- `IO_DO_35`

Furthermore they provide an over-current monitoring to detect overload situations and analog (ADC) feedback.

For details on the usage refer to the **driver functions for digital inputs and outputs**.

PVG/Voltage Outputs

PVG Outputs (primary function)

These pins can be used as digital low-side switches:

- `IO_PVG_00`
- `IO_PVG_01`
- `IO_PVG_02`
- `IO_PVG_03`
- `IO_PVG_04`
- `IO_PVG_05`

For details on the usage refer to the **driver functions for PVG Outputs**.

As a secondary function they can be used as:

- **Analog Inputs**
- **Digital Inputs**
- **Digital Outputs**

Voltage Outputs (secondary function of PVG Outputs)

These pins can be used as voltage outputs:

- `IO_VOUT_00`
- `IO_VOUT_01`
- `IO_VOUT_02`
- `IO_VOUT_03`
- `IO_VOUT_04`
- `IO_VOUT_05`

Furthermore they analog (ADC) feedback.

These outputs are driven by a configurable PID controller.

For details on the usage refer to the **driver functions for voltage outputs**.

Pin and diagnostic features

Explicit overview of the diagnostic functions of the ECU pins.

Diagnostic features of HY-TTC 30 Family

PIN GROUP			FEATURES	DIAGNOSTIC FUNCTION	
Group	Pin Numbers	Defines	I/O Features	Diagnostics	Error Code
PWM output groups					
PWM Output with timer feedback, analog feedback and current feedback	IO_PIN_H1 IO_PIN_G1 IO_PIN_F1 IO_PIN_E1 IO_PIN_D1 IO_PIN_C1	IO_PWM_00 IO_PWM_01 IO_PWM_02 IO_PWM_03 IO_PWM_04 IO_PWM_05	PWM Output	open load detection	IO_E_PWM_OPEN_LOAD
				short circuit to GND detection	IO_E_PWM_SHORT_CIRCUIT
				short circuit to UBAT detection	IO_E_PWM_SHORT_BATTER
				the PWM output is disabled	IO_E_PWM_OUTPUT_DISAB
				measured signal period is too small	IO_E_PWM_CAPTURE_ERRC
				Current monitoring and over current protection. Be reminded that the output protection will return these errors not only if the output current is too high but if one of the following conditions are met: - over temperature - Supply for power stages disconnected (due to loss of gate drive for reverse polarity protection)	IO_E_PROT_USER_OVERLO IO_E_PROT_TEMP_OVERLO IO_E_PROT_ACTIVE IO_E_PROT_FATAL IO_E_PROT_REENABLE
		IO_DO_00 IO_DO_01	Digital Output	open load detection	IO_E_DO_OPEN_LOAD

IO_DO_02 IO_DO_03 IO_DO_04 IO_DO_05			short circuit to GND detection	IO_E_DO_SHORT_CIRCUIT
			short circuit to UBAT detection	IO_E_DO_SHORT_BATTERY
			open load detection / short circuit to UBAT detection (if a pull up resistor is configured)	IO_E_DO_OPEN_LOAD_OR_
			low pass of digital outputs with analog feedback is being tuned in.	IO_E_DO_DIAG_TRANSIENT
			Current monitoring and current protection. Be reminded that the output protection will return these errors not only if the output current is too high but if one of the following conditions are met: - over temperature - Supply for power stages disconnected (due to loss of gate drive for reverse polarity protection)	IO_E_PROT_USER_OVERLO IO_E_PROT_TEMP_OVERLO IO_E_PROT_ACTIVE IO_E_PROT_FATAL IO_E_PROT_REENABLE
IO_DO_20 IO_DO_21 IO_DO_22 IO_DO_23 IO_DO_24 IO_DO_25	Digital Output		open load detection	IO_E_DO_OPEN_LOAD
			short circuit to GND detection	IO_E_DO_SHORT_CIRCUIT
			short circuit to	IO_E_DO_SHORT_BATTERY

			UBAT detection	
			open load detection / short circuit to UBAT detection (if a pull up resistor is configured)	IO_E_DO_OPEN_LOAD_OR_
			low pass of digital outputs with analog feedback is being tuned in.	IO_E_DO_DIAG_TRANSIENT
			Current monitoring and current protection. Be reminded that the output protection will return these errors not only if the output current is too high but if one of the following conditions are met: - over temperature - Supply for power stages disconnected (due to loss of gate drive for reverse polarity protection)	IO_E_PROT_USER_OVERLO IO_E_PROT_TEMP_OVERLO IO_E_PROT_ACTIVE IO_E_PROT_FATAL IO_E_PROT_REENABLE
		IO_ADC_34 IO_ADC_35 IO_ADC_36 IO_ADC_37 IO_ADC_38 IO_ADC_39	Analog Input	no built-in diagnostic functions.
		IO_DI_24 IO_DI_25 IO_DI_26 IO_DI_27	Digital Input (with proper limits configured)	open load detection / short circuit to GND IO_E_DI_OPEN_LOAD_OR_S

		IO_DI_28 IO_DI_29		detection (if a pull down resistor is configured)	
				short circuit to UBAT detection	IO_E_DI_SHORT_BATTERY
				invalid voltage level detection (i.e. outside of specified limits)	IO_E_DI_INVALID_VOLTAGE
				short to GND detection	IO_E_DI_SHORT_CIRCUIT
				open load detection	IO_E_DI_OPEN_LOAD
		IO_PWD_20 IO_PWD_21 IO_PWD_10 IO_PWD_11 IO_PWD_12 IO_PWD_13	Digital Timer Input	measured signal period is too small	IO_E_PWD_CAPTURE_ERROR
				time measurement not finished yet	IO_E_PWD_NOT_FINISHED
				only a constant high level is detected	IO_E_PWD_HIGH_LEVEL
				only a constant low level is detected	IO_E_PWD_LOW_LEVEL
PWM Output with timer feedback, analog feedback	IO_PIN_K1 IO_PIN_J1	IO_PWM_10 IO_PWM_11	PWM Output	open load detection	IO_E_PWM_OPEN_LOAD
				short circuit to GND detection	IO_E_PWM_SHORT_CIRCUIT
				short circuit to UBAT detection	IO_E_PWM_SHORT_BATTERY
				the PWM output is disabled	IO_E_PWM_OUTPUT_DISABLED
				measured signal period is too small	IO_E_PWM_CAPTURE_ERROR
				Over current monitoring and over current protection.	IO_E_PROT_USER_OVERLOAD IO_E_PROT_TEMP_OVERLOAD IO_E_PROT_ACTIVE IO_E_PROT_FATAL IO_E_PROT_REENABLE

			Be reminded that the output protection will return these errors not only if the output current is too high but if one of the following conditions are met: - over temperature - Supply for power stages disconnected (due to loss of gate drive for reverse polarity protection)	
IO_DO_06 IO_DO_07	Digital Output	open load detection	IO_E_DO_OPEN_LOAD	
		short circuit to GND detection	IO_E_DO_SHORT_CIRCUIT	
		short circuit to UBAT detection	IO_E_DO_SHORT_BATTERY	
		open load detection / short circuit to UBAT detection (if a pull up resistor is configured)	IO_E_DO_OPEN_LOAD_OR_	
		low pass of digital outputs with analog feedback is being tuned in.	IO_E_DO_DIAG_TRANSIENT	
		Over current protection. Be reminded that the output protection will return these errors not only if the output	IO_E_PROT_USER_OVERLO IO_E_PROT_TEMP_OVERLO IO_E_PROT_ACTIVE IO_E_PROT_FATAL IO_E_PROT_REENABLE	

				current is too high but if one of the following conditions are met: - over temperature - Supply for power stages disconnected (due to loss of gate drive for reverse polarity protection)	
		IO_ADC_40 IO_ADC_41	Analog Input	no built-in diagnostic functions.	
		IO_DI_30 IO_DI_31	Digital Input (with proper limits configured)	open load detection / short circuit to GND detection (if a pull down resistor is configured)	IO_E_DI_OPEN_LOAD_OR_S
				short circuit to UBAT detection	IO_E_DI_SHORT_BATTERY
				invalid voltage level detection (i.e. outside of specified limits)	IO_E_DI_INVALID_VOLTAGE
				short to GND detection	IO_E_DI_SHORT_CIRCUIT
				open load detection	IO_E_DI_OPEN_LOAD
		IO_PWD_22 IO_PWD_23	Digital Timer Input	measured signal period is too small	IO_E_PWD_CAPTURE_ERRC
				time measurement not finished yet	IO_E_PWD_NOT_FINISHED
				timer overflow occurred	IO_E_PWD_TIMER_OVERFLC
		Digital output groups			
Digital Output	IO_PIN_B1 IO_PIN_A1	IO_DO_10 IO_DO_11	Digital Output	open load detection	IO_E_DO_OPEN_LOAD

low-side with analog feedback			short circuit to GND detection	IO_E_DO_SHORT_CIRCUIT
			short circuit to UBAT detection	IO_E_DO_SHORT_BATTERY
			open load detection / short circuit to UBAT detection (if a pull up resistor is configured)	IO_E_DO_OPEN_LOAD_OR_
			low pass of digital outputs with analog feedback is being tuned in.	IO_E_DO_DIAG_TRANSIENT
			Over current protection. Be reminded that the output protection will return these errors not only if the output current is too high but if one of the following conditions are met: - Over temperature - Supply for power stages disconnected (due to loss of gate drive for reverse polarity protection) - OVP circuitry has been tripped - External watchdog has disabled the output stages	IO_E_PROT_USER_OVERLO IO_E_PROT_TEMP_OVERLO IO_E_PROT_ACTIVE IO_E_PROT_FATAL IO_E_PROT_REENABLE
	IO_ADC_28	Analog	no built-in	

		IO_ADC_29	Input	diagnostic functions.	
		IO_DI_22 IO_DI_23	Digital Input (with proper limits configured)	open load detection	IO_E_DI_OPEN_LOAD
				short circuit to UBAT detection	IO_E_DI_SHORT_BATTERY
				short circuit to GND detection	IO_E_DI_SHORT_CIRCUIT
				invalid voltage level detection (i.e. outside of specified limits)	IO_E_DI_INVALID_VOLTAGE
				open load / short circuit to GND detection (if a pull down resistor is configured)	IO_E_DI_OPEN_LOAD_OR_S
Analog input groups					
4-Mode ADC	IO_PIN_J4 IO_PIN_H4	IO_ADC_00 IO_ADC_01	Analog Input	configuration switch protection	IO_E_FET_PROTECTION
		IO_LED_00 IO_LED_01	LED Switch	an over current was detected	IO_E_FET_PROTECTION
		IO_DI_00 IO_DI_01	Digital Input (with proper limits configured)	open load detection / short circuit to GND detection (if a pull down resistor is configured)	IO_E_DI_OPEN_LOAD_OR_S
				short circuit to UBAT detection	IO_E_DI_SHORT_BATTERY
				invalid voltage level detection (i.e. outside of specified limits)	IO_E_DI_INVALID_VOLTAGE
				short to GND detection	IO_E_DI_SHORT_CIRCUIT
				open load detection	IO_E_DI_OPEN_LOAD
3-Mode	IO_PIN_E4	IO_ADC_10	Analog	configuration	IO_E_FET_PROTECTION

ADC	IO_PIN_D4 IO_PIN_C4 IO_PIN_B4 IO_PIN_A4 IO_PIN_A3	IO_ADC_11 IO_ADC_12 IO_ADC_13 IO_ADC_14 IO_ADC_15	Input	switch protection	
		IO_LED_02 IO_LED_03 IO_LED_04 IO_LED_05 IO_LED_06 IO_LED_07	LED Switch	an over current was detected	IO_E_FET_PROTECTION
		IO_DI_10 IO_DI_11 IO_DI_12 IO_DI_13 IO_DI_14 IO_DI_15	Digital Input (with proper limits configured)	open load detection / short circuit to GND detection (if a pull down resistor is configured)	IO_E_DI_OPEN_LOAD_OR_S
				short circuit to UBAT detection	IO_E_DI_SHORT_BATTERY
				invalid voltage level detection (i.e. outside of specified limits)	IO_E_DI_INVALID_VOLTAGE
				short to GND detection	IO_E_DI_SHORT_CIRCUIT
				open load detection	IO_E_DI_OPEN_LOAD
1-Mode ADC	IO_PIN_G4 IO_PIN_G4	IO_ADC_20 IO_ADC_21	Analog Input	no built-in diagnostic functions.	
		IO_DI_02 IO_DI_03	Digital Input (with proper limits configured)	open load detection / short circuit to GND detection (if a pull down resistor is configured)	IO_E_DI_OPEN_LOAD_OR_S
				short circuit to UBAT detection	IO_E_DI_SHORT_BATTERY
				invalid voltage level detection (i.e. outside of specified limits)	IO_E_DI_INVALID_VOLTAGE
				short to GND	IO_E_DI_SHORT_CIRCUIT

				detection	IO_E_DI_OPEN_LOAD	
				open load detection		
Digital timer input groups						
Digital Timer Input	IO_PIN_E3 IO_PIN_D3 IO_PIN_C3 IO_PIN_B3	IO_PWD_00 IO_PWD_01 IO_PWD_02 IO_PWD_03	Digital Timer Input	measured signal period is too small	IO_E_PWD_CAPTURE_ERRC	
				time measurement not finished yet	IO_E_PWD_NOT_FINISHED	
				timer overflow occurred	IO_E_PWD_TIMER_OVERFLC	
				only a constant high level is detected	IO_E_PWD_HIGH_LEVEL	
				only a constant low level is detected	IO_E_PWD_LOW_LEVEL	
		IO_ADC_30 IO_ADC_31 IO_ADC_32 IO_ADC_33	Analog Input	no built-in diagnostic functions.		
		IO_DI_04 IO_DI_05 IO_DI_06 IO_DI_07	Digital Input (with proper limits configured)	open load detection / short circuit to GND detection (if a pull down resistor is configured)	IO_E_DI_OPEN_LOAD_OR_S	
				short circuit to UBAT detection	IO_E_DI_SHORT_BATTERY	
				invalid voltage level detection (i.e. outside of specified limits)	IO_E_DI_INVALID_VOLTAGE	
				short to GND detection	IO_E_DI_SHORT_CIRCUIT	
	open load detection			IO_E_DI_OPEN_LOAD		
	PVG output groups					
	PVG Output	IO_PIN_K2 IO_PIN_J2 IO_PIN_H2 IO_PIN_G2	IO_PVG_00 IO_PVG_01 IO_PVG_02 IO_PVG_03	PVG Output	short circuit to UBAT detection	IO_E_PVG_SHORT_BATTER'
					short circuit to	IO E PVG SHORT CIRCUIT

IO_PIN_F2 IO_PIN_E2	IO_PVG_04 IO_PVG_05		GND detection	
			the PVG output is disabled	IO_E_PVG_OUTPUT_DISABL
			over current protection	IO_E_PROT_ACTIVE IO_E_P
	IO_VOUT_00 IO_VOUT_01 IO_VOUT_02 IO_VOUT_03 IO_VOUT_04 IO_VOUT_05	Voltage Output	short circuit to UBAT detection	IO_E_VOUT_SHORT_BATTEI
			short circuit to GND detection	IO_E_VOUT_SHORT_CIRCUI
			the configured output voltage was not reached within settling time	IO_E_VOUT_PRECISION
			the voltage output is disabled	IO_E_VOUT_OUTPUT_DISAE
			over current protection	IO_E_PROT_ACTIVE IO_E_P
	IO_ADC_22 IO_ADC_23 IO_ADC_24 IO_ADC_25 IO_ADC_26 IO_ADC_27	Analog Input	no built-in diagnostic functions.	
	IO_DI_16 IO_DI_17 IO_DI_18 IO_DI_18 IO_DI_19 IO_DI_20 IO_DI_21	Digital Input (with proper limits configured)	open load detection / short circuit to GND detection (if a pull down resistor is configured)	IO_E_DI_OPEN_LOAD_OR_S
			short circuit to UBAT detection	IO_E_DI_SHORT_BATTERY
			invalid voltage level detection (i.e. outside of specified limits)	IO_E_DI_INVALID_VOLTAGE
			short to GND detection	IO_E_DI_SHORT_CIRCUIT
			open load detection	IO_E_DI_OPEN_LOAD

		IO_DO_30 IO_DO_31 IO_DO_32 IO_DO_33 IO_DO_34 IO_DO_35	Digital Output	short circuit to UBAT detection	IO_E_DO_SHORT_BATTERY
				short circuit to GND detection	IO_E_DO_SHORT_CIRCUIT
				the digital output is disabled	IO_E_DO_OUTPUT_DISABLE
				over current protection	IO_E_PROT_ACTIVE IO_E_P



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

[Main Page](#)[Related Pages](#)[Data Structures](#)[Files](#)

ECU Map

Description and properties of the ECU

ECU properties

The HY-TTC 30 ECU is equipped with an Infineon XC2267 microcontroller which is driven at a CPU frequency of 80MHz.

Memory Map

The HY-TTC 30 provides different memory types. The following table shows which memories and sizes are available:

Memory	Size
EEPROM	8128 Byte
internal Flash	768 kByte
internal RAM	82 kByte

Generated on Mon Nov 16 2020 16:59:48 for HY-TTC 30 Family C API Manual by

doxygen 1.8.2



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

[Main Page](#)[Related Pages](#)[Data Structures](#)[Files](#)

Implementation Examples for Safety Functions

Example for using the IO-Driver in a safety critical environment

Example implementation for Safety-Callback

See **DIAG_ERR_CALLBACK** for details on the safety callback function.

For further details on error types and how they are treated by the diagnostic state machine, refer to **Diagnostic state machine error codes**.

```
static ubyte1 APPL_SafetyCb( ubyte1 diag_state
                             , DIAG_ERRORCODE *
                             const error )
{
    ubyte1 action;

    // Error codes can for example be saved to
    // error memory of application software
    // (APPL_WriteErrMem is a hypothetical
    // function of the application software).
    APPL_WriteErrMem( diag_state, error );

    // Just a simple example which does
    // not take the type of error into account
    switch (error->device_num)
    {
        case IO_ADC_00:
        case IO_ADC_01:
            // ...
            // Do something sophisticated
            // ...
            action = DIAG_ERR_NOACTION;
            break;
        case IO_PWM_00:
        case IO_PWM_01:
```

```
case IO_PWM_02:
case IO_PWM_03:
    // ...
    // Do something sophisticated
    // ...
    action = DIAG_ERR_SAFESTATE;
    break;
case IO_PWM_04:
case IO_PWM_05:
case IO_PWM_10:
case IO_PWM_11:
    // ...
    // Do something sophisticated
    // ...
    action = DIAG_ERR_NOACTION;
    break;
default:
    action = DIAG_ERR_SAFESTATE;
    break;
}

return action;
}
```

Example for calling PWM step functions for safety critical PWM outputs to explain how to react on certain error codes of the step functions:

```
void APPL_PWM_ControlOutput( void )
{
    bool pwm_current_new_00;
    ubyte2 pwm_current_00;
    ubyte2 duty;
    IO_ErrorType rc_pwm_00;
    IO_ErrorType rc_pwm_current_00;

    rc_pwm_current_00 = IO_PWM_GetCur(
        IO_PWM_00, &pwm_current_00,
        &pwm_current_new_00 );
    // Application controller for PWM
    (hypothetical user function)
    duty = APPL_PWM_PidCtrl( IO_PWM_00,
        pwm_current_00, pwm_current_new_00,
        rc_pwm_current_00 );

    // set duty cycle
    rc_pwm_00 = IO_PWM_SetDuty( IO_PWM_00,
        duty, NULL );

    switch (rc_pwm_00)
    {
        case IO_E_OK:
            // Everything fine
            break;
        case IO_E_PWM_SHORT_CIRCUIT:
            // Application code
            break;
    }
}
```

```
        case
IO_E_PWM_OPEN_LOAD_OR_SHORT_BATTERY:
    // Application code
    break;
    case IO_E_PWM_SHORT_BATTERY:
    // Application code
    break;
    case IO_E_PWM_OPEN_LOAD:
    // Application code
    break;
    case ...
    // Application code
    break;
    default:
    // Unexpected error code
    break;
}
}
```

Example for safety critical Driver configuration

```
// Safety configuration for IO-Driver
static const IO_DRIVER_SAFETY_CONF
    c_driver_safety_conf =
{
    .safety_switch_type =
    IO_DRIVER_SAFETY_SWITCH_INT
    , .glitch_filter_time = 40      // [ms]
    , .cycle_time = 11000          // [us] -
    > task cycle time plus tolerance
    , .error_callback = &APPL_SafetyCb //
    (Set to NULL if IO-Driver should decide
    what to do on errors)
};

// Safety configuration for PWM Output
static const IO_PWM_SAFETY_CONF
    c_pwm_safety_conf =
{
    IO_SAFETY_SWITCH_0
    , 80
    , 100
    , 200
};

void task (void)
{
    IO_ErrorType rc_driv_begin;
    IO_ErrorType rc_driv_end;

    rc_driv_begin = IO_Driver_TaskBegin();
    if (rc_driv_begin != IO_E_OK)
    {
```

```

        // User code
    }

    // User Application
    // and calls to driver task functions.
    APPL_PWM_ControlOutput();

    rc_driv_end = IO_Driver_TaskEnd();
    if (rc_driv_end != IO_E_OK)
    {
        // User code
    }
}

void main (void)
{
    ubyte4 timestamp;
    IO_ErrorType rc_driv_init;
    IO_ErrorType rc_pwm_init;

    //-----//
    // start of driver initialization //
    //-----//

    // IO_Driver_Init() is the first function:
    rc_driv_init = IO_Driver_Init(
        IO_DRIVER_MODE_SERVICE_WD,
        &c_driver_safety_conf ); // safety
    critical application
    if (rc_driv_init != IO_E_OK)
    {
        // User code
    }

    // Initialization of PWM Output, 200Hz
    rc_pwm_init = IO_PWM_Init( IO_PWM_00, 200,
        TRUE, TRUE, 0, &c_pwm_safety_conf);
    if (rc_pwm_init != IO_E_OK)

```

```
{  
    // User code  
}  
  
//-----//  
// end of driver initialization //  
//-----//  
  
//-----  
---//  
// from now on only task functions are  
called //  
//-----  
---//  
while (1){  
    IO_RTC_StartTime(&timestamp);  
    task();  
    while (IO_RTC_GetTimeUS(timestamp) <  
10000);  
}  
}
```

Example for manual safety switch test

This example shows how to perform a manual safety switch test (for both internal and external safety switch configuration), which is necessary in case safety critical configured PWM outputs are applied with loads **after** the **ECU startup**.

```
// number of safety configured PWM outputs //
#define SAFETY_PWM_NUM 2
// number of software cycle to elapse before
    the return value from IO_PWM_SetDuty is
    plausible //
#define SSW_CYCLES 5

// status for safety critical PWM outputs //
typedef struct pwm_status
{
    IO_PIN pwm_pin;
    IO_ErrorType pwm_set_rc;
    ubyte2 duty_cycle;

    // flag that indicates if the respective
    PMW output has to be tested with a manual
    safety switch test again //
    bool pwm_man_ssw_test;
} PWM_STATUS;

// states of manual safety switch test //
typedef enum { SSW_START,
               SSW_DISABLED,
               SSW_ENABLED,
               SSW_WAIT,
               SSW_DONE
} SSW_STATE;

void Man_SSW_Test (IO_PIN safety_switch);
```

```

static IO_ErrorType driver_init_rc;
static IO_ErrorType driver_task_begin_rc;
static IO_ErrorType driver_task_end_rc;
static IO_ErrorType pwm_init_rc;
static IO_ErrorType power_rc;
static IO_ErrorType diag_rc;
static DIAG_ERRORCODE diag_error;
static ubyte1 diag_state;

// insert ALL safety configured PWM outputs in
// this structure! //
// set duty cycle only via:
    (pwm_status[i].duty_cycle = value)! //
static PWM_STATUS pwm_status[] = {{IO_PIN_H1,
    IO_E_OK, 0x8000, FALSE },
                                     {IO_PIN_G1,
    IO_E_OK, 0x8000, FALSE }};
                                     // ...

// global variables for manual safety switch
// test //
static SSW_STATE ssw_test_next_state =
    SSW_START;
static SSW_STATE ssw_test_prev_state =
    SSW_START;
static ubyte1 ssw_cnt = 0;

static ubyte1 APPL_SafetyCb( ubyte1 diag_state,
    DIAG_ERRORCODE * const error )
{
    ubyte1 i;

    switch (error->device_num )
    {
        // add case for ALL safety configured
        // PWM output pins! //
        case IO_PIN_H1:

```

```

        case IO_PIN_G1:
        // ...
            switch ( error->error_code )
            {
                // ignore error case and label
                the affected PWM output for later manual
                ssw testing (pwm_man_ssw_test) //
                case DIAG_E_SAFETY_SW_SHUT_OFF:
                    for (i = 0; i <
SAFETY_PWM_NUM; i++)
                    {
                        if
                        (pwm_status[i].pwm_pin == error-
>device_num)
                            {

                                pwm_status[i].pwm_man_ssw_test = TRUE;
                                }
                            }
                        return DIAG_ERR_NOACTION;
                    default:
                        return DIAG_ERR_NOACTION;
                }
            default:
                return DIAG_ERR_SAFESTATE;
        }
    }

// safety configuration for IO-Driver //
static const IO_DRIVER_SAFETY_CONF
    driver_safety_conf =
{
    IO_DRIVER_SAFETY_SWITCH_INT, // or
    IO_DRIVER_SAFETY_SWITCH_EXT,
    180,
    11000,

```

```

    APPL_SafetyCb
};

// safety configuration for PWM output //
static const IO_PWM_SAFETY_CONF pwm_safety_conf
=
{
    IO_SAFETY_SWITCH_0, // or
    IO_SAFETY_SWITCH_1
    50,
    150,
    200
};

void main (void)
{
    ubyte1 i;
    ubyte4 timestamp;

    driver_init_rc = IO_Driver_Init (
        IO_DRIVER_MODE_DEFAULT, &driver_safety_conf
    );

    // safety configured PWM outputs
    pwm_init_rc = IO_PWM_Init( IO_PIN_H1, 200,
        TRUE, TRUE, 5000, &pwm_safety_conf );
    pwm_init_rc = IO_PWM_Init( IO_PIN_G1, 200,
        TRUE, TRUE, 5000, &pwm_safety_conf );
    // ...

    power_rc = IO_POWER_Set(
        IO_INT_POWERSTAGE_ENABLE, IO_POWER_ON);

    while (1)
    {
        (void) IO_RTC_StartTime (&timestamp);
        driver_task_begin_rc =
        IO_Driver_TaskBegin();
    }
}

```

```

// User code //

// ===== SAFETY PWM
OUTPUTS ===== //

// ensure that the duty cycle of not
yet tested safety PWM outputs is 0%!//
for (i = 0; i < SAFETY_PWM_NUM; i++)
{
    if (pwm_status[i].pwm_man_ssw_test
!= FALSE)
    {
        pwm_status[i].pwm_set_rc =
IO_PWM_SetDuty(pwm_status[i].pwm_pin,

0,

NULL );
    }
    else
    {
        pwm_status[i].pwm_set_rc =
IO_PWM_SetDuty(pwm_status[i].pwm_pin,

pwm_status[i].duty_cycle,

NULL );
    }
}

// trigger manual safety switch test
(only during diagnostic main state!) //
// -----
----- //

```

```

        // Note: As the function Man_SSW_Test
test has to be called several times //
        // in order to perform a complete
manual safety switch test, it must be    //
        // ensured that the user defined
trigger condition does not inhibit the    //
        // function call, once the manual
safety switch test has been started!    //
        // -----
----- //
        if ( (diag_state == DIAG_STATE_MAIN) &&
            (( #INSERT TRIGGER CONDITION HERE#
) || ( ssw_test_next_state != SSW_START ) )
        )
        {
            Man_SSW_Test(
pwm_safety_conf.safety_switch );
        }

        //
=====
===== //

        // User code //

        // retrieve diagnostic state and error
code //
        diag_rc = DIAG_Status(&diag_state,
&diag_error);

        driver_task_end_rc = IO_Driver_TaskEnd
();
        while (IO_RTC_GetTimeUS (timestamp) <
10000);
    }

```

```

}

// ===== MANUAL SAFETY SWITCH
// TEST ===== //
void Man_SSW_Test (IO_PIN safety_switch)
{
    ubyte1 i;

    switch (ssw_test_next_state)
    {
        // [1]: start manual safety switch test
        //
        case SSW_START:
            (void) IO_POWER_Set( safety_switch,
            IO_POWER_OFF);
            ssw_test_next_state = SSW_WAIT;
            ssw_test_prev_state = SSW_START;
            break;

            // [3]: check PWM set error only for
            not yet tested PWM outputs with load after
            SSW is disabled //
            case SSW_DISABLED:
                for (i = 0; i < SAFETY_PWM_NUM;
                i++)
                {
                    if
                    ((pwm_status[i].pwm_man_ssw_test != FALSE)
                    &&
                    (pwm_status[i].pwm_set_rc
                    != IO_E_PWM_OPEN_LOAD))
                    {
                        // Error: PWM output is not
                        correctly disabled! //
                        if
                        (pwm_status[i].pwm_set_rc == IO_E_OK)
                        {

```

```

                (void)
DIAG_EnterSafestate();
            }
        }
    }
    (void) IO_POWER_Set( safety_switch,
IO_POWER_ON);
    ssw_test_next_state = SSW_WAIT;
    ssw_test_prev_state = SSW_DISABLED;
    break;

    // [5]: check PWM set error only for
not yet tested PWM outputs with load after
SSW is enabled //
    case SSW_ENABLED:
        for (i = 0; i < SAFETY_PWM_NUM;
i++)
        {
            if
((pwm_status[i].pwm_man_ssw_test != FALSE)
&&
                (pwm_status[i].pwm_set_rc
!= IO_E_PWM_OPEN_LOAD))
            {
                // Error: PWM output is not
correctly enabled! //
                if
(pwm_status[i].pwm_set_rc != IO_E_OK)
                {
                    (void)
DIAG_EnterSafestate();
                }
            }
        }
    }
    (void) IO_POWER_Set( safety_switch,
IO_POWER_ON);

```

```

        ssw_test_next_state = SSW_DONE;
        break;

        // [6]: set the tested flag for
        affected PWM outputs with load //
        case SSW_DONE:
            for (i = 0; i < SAFETY_PWM_NUM;
i++)
            {
                if
                ((pwm_status[i].pwm_man_ssw_test != FALSE)
                &&
                (pwm_status[i].pwm_set_rc
                != IO_E_PWM_OPEN_LOAD))
                {

pwm_status[i].pwm_man_ssw_test = FALSE;
                }
            }
            ssw_test_next_state = SSW_START;
            break;

            // [2, 4]: wait some software cycles,
            in order to receive plausible return values
            from IO_PWM_SetDuty() //
            case SSW_WAIT:
                if (ssw_cnt >= SSW_CYCLES)
                {
                    if (ssw_test_prev_state ==
SSW_START)
                    {
                        ssw_test_next_state =
SSW_DISABLED;
                    }
                    else
                    {

```

```
        ssw_test_next_state =  
SSW_ENABLED;  
    }  
    ssw_cnt = 0;  
}  
else  
{  
    ssw_cnt++;  
}  
break;  
}  
}
```



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

[Main Page](#)[Related Pages](#)[Data Structures](#)[Files](#)

Examples for using UDS support functions

This chapter shows some examples how to use the support functions for UDS reprogramming.

Read DataIdentifiers stored in the Branding Block

This section describes how to read a DataIdentifier that has been stored in the ECU's branding block.

```
IO_ErrorType rc_validate;
IO_ErrorType rc_get_did;
IO_BRBL_CUSTOM_DID did_item;

// First the consistency of the BRBL has to be
// checked. The validity of
// the BRBL has to be checked only once per
// power cycle. It is best to
// call this function during the init phase
// (directly after calling \c IO_Driver_Init)
// because validating the BRBL will take
// approximately 1.2ms.
rc_validate = IO_BRBL_Validate();
if (rc_validate == IO_E_OK)
{
    rc_get_did = IO_BRBL_GetDid(
        IO_BRBL_CUSTOM_DID_IDX_0, &did_item );
    if (rc_get_did == IO_E_OK)
    {
        // The DID data can now be accessed as
        // follows:
        // did_item.Did holds the DID (e.g.
        // 0xFD00)
        // did_item.DidLength holds the length
        // of the data that is attached to the DID
        // did_item.DidData is a pointer to the
        // DID data (array with length
        // did_item.DidLength)
```

```
        // Copying the DID data from the
        branding block can be done as follows:
        for (i = 0; i < did_item.DidLength;
i++)
        {
            app_did_data_buffer[i] =
did_item.DidData[i];
        }
    }
else
{
    // Reading from the BRBL is not possible
    because it could not be validated.
}
```

Machine active - Reprogramming not allowed

If the machine is currently active, reprogramming is not allowed in general. In such a case none of the driver functions for UDS support may be called. The following response shall be sent by the application:

```
//Tester: 02 10 02 55 55 55 55 55 (Request to  
        switch to the programming sessions)  
//App:    03 7F 10 22 55 55 55 55 (Request  
        denied with code "conditions not correct")
```

Reset to boot mode with authentication in bootloader

Example for giving control to the bootloader and letting the bootloader handle the security access. In the example the length of the seed/key mechanism has been assumed to be 4 bytes (setting in branding block). This method (giving control to the bootloader triggered by the session control) is the one suggested within ISO 14229-1:2013. It is also the easiest solution to implement for the application software.

```
// Tester: 02 10 02 55 55 55 55 55 (App checks
//         if machine is inactive ...)
// If reprogramming is allowed in the current
//         operating state the application
// gives control to the bootloader. Please be
//         aware that a response pending
// has to be sent in case the application does
//         not switch to boot mode immediately
// but needs some time to shut down to a safe
//         operating state!
(void)IO_Driver_ResetToBootMode(
    IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_BL
    ,
    IO_DRIVER_RTBM_UDS_RSP_SEND );
// The bootloader will now take over. Note: In
//         this case the bootloader will send the
//         positive response
// to the request to switch to the programming
//         session (see function \c
//         IO_Driver_ResetToBootMode).
// BL      : 06 50 02 00 32 01 F4 55
// Tester: 02 27 07 55 55 55 55 55
// BL      : 06 67 07 xx xx xx xx 55
```

```
// Tester: 06 27 08 yy yy yy yy 55  
// BL      : 02 67 08 55 55 55 55 55
```

Reset to boot mode with authentication in application

Example for giving control to the bootloader after the application software handled the security access. In the example the length of the seed/key mechanism has been assumed to be 4 bytes (setting in branding block).

Attention

With this approach the authentication within the bootloader is bypassed and has to be handled by the application. The responsibility for correct authentication lies solely with the application and therefore with the system integrator.

```
ubyte4 prn;
ubyte4 key;
ubyte4 secret_key[IO_BRBL_XTEA_PRIV_KEY_LEN];

// Tester: 02 10 02 55 55 55 55 55 (App checks
//      if machine is inactive ...)
// App    : 06 50 02 00 32 01 F4 55 (... if yes
//      a positive response needs to be sent.)
// Tester: 02 27 07 55 55 55 55 55
// App    : Calls the driver function to
//      generate a random seed.
(void)IO_Crypt_GetPseudoRandomNumber( &prn );
//Be aware that service 0x27 uses the Big
//      Endian format but the XC2000 CPU uses the
//      Little
//Endian format. Thus when copying prn to the
//      UDS message, byte3 of prn must be on the
//      LSB position
//in the response message and byte0 of prn on
//      the MSB position.
```

```

// App    : 06 67 07 xx xx xx xx 55 (Random Seed
            "xx xx xx xx" from
            IO_Crypt_GetPseudoRandomNumber)
// Tester: 06 27 08 yy yy yy yy 55
// App    : Deciphers the key sent by the tester
            by passing "yy yy yy yy" to the XTEA
            decipher function.
// Be aware that the key is sent by the tester
            in Big Endian format. It must be stored in
            the variable
// key in Little Endian format.
(void)IO_BRBL_GetXteaKey(
    IO_BRBL_XTEA_PRIV_KEY_IDX_1
                                , secret_key //array of
    IO_BRBL_XTEA_PRIV_KEY_LEN elements
                                ,
    IO_BRBL_XTEA_PRIV_KEY_LEN );
(void)IO_Crypt_XteaDecipher32( &key //yyyyyyyyy
                                , secret_key
                                , TRUE );
// The key is now deciphered and must match the
            sent seed again (otherwise a negative
            response has to be sent!).
// If seed matches the deciphered key the
            control can be given to the bootloader.
IO_Driver_ResetToBootMode(
    IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_APP
                                ,
    IO_DRIVER_RTBM_UDS_RSP_SEND );
// The bootloader will now take over. Note: In
            this case the bootloader will send the
            positive response
// to the previous service 0x27 message of the
            tester (see function \c
            IO_Driver_ResetToBootMode).
// BL     : 02 67 08 55 55 55 55 55







```


Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Structures

Here are the data structures with brief descriptions:

G _diag_errorcode	Diagnostic Error code structure
G _io_adc_safety_conf	Safety configuration for the ADC inputs
G _io_brbl_can_id_	CAN ID structure
G _io_brbl_can_param	Branding block CAN parameter structure
G _io_brbl_dids	Entry definition for DID table
G _io_can_data_frame	CAN data frame
G _io_driver_di_limits	Voltage limits for digital inputs
G _io_driver_rst_info	Reset information
G _io_driver_safety_conf	Driver Safety Configuration
G _io_driver_trap_info	Contains information regarding traps/exceptions
G _io_pid_config	PID configuration structure
G _io_pwd_cplx_safety_conf	Safety configuration for the Complex PWD inputs
G _io_pwd_inc_safety_conf	Safety configuration for the Incremental or Counter

 _io_pwd_pulse_samples	PWD inputs PWD pulse-width data structure
 _io_pwm_current_queue	PWM current measurement queue
 _io_pwm_current_safety_conf	Safety configuration for the PWM outputs
 _io_pwm_safety_conf	Safety configuration for the PWM outputs
 ApdbType	APDB structure
 can_id	CAN ID structure

Generated on Mon Nov 16 2020 16:59:48 for HY-TTC 30 Family C API Manual by

 1.8.2

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_diag_errorcode **Struct Reference**

Diagnostic Error code structure. [More...](#)

```
#include <DIAG_Constants.h>
```

Data Fields

DIAG_ErrorType	error_code
-----------------------	-------------------

IO_PIN	device_num
---------------	-------------------

ubyte2	faulty_value
---------------	---------------------

Detailed Description

Diagnostic Error code structure.

Stores all relevant error parameters returned from the diagnostic state machine or returned from the WD

Field Documentation

IO_PIN _diag_errorcode::device_num

device number which caused error

DIAG_ErrorType _diag_errorcode::error_code

error code: for a detailed description of the diagnostic error codes see [Listing of diagnostic errors](#)

ubyte2 _diag_errorcode::faulty_value

value which caused the error

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

- [DIAG_Constants.h](#)

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_adc_safety_conf Struct Reference

Safety configuration for the ADC inputs. [More...](#)

```
#include <IO_ADC.h>
```

Data Fields

ubyte2 **adc_val_upper**

ubyte2 **adc_val_lower**

Detailed Description

Safety configuration for the ADC inputs.

Stores all relevant safety configuration parameters for the ADC inputs as an absolute value. The values shall be stated as mV, uA or Ohm - depending on the configuration of the channel.

It is not allowed to set the lower limit higher than or equal to the upper limit.

The valid min/max limits for each measured unit and range are like follows:

- `IO_ADC_ABSOLUTE` for 5V range configuration: 1mV ... 5000mV
- `IO_ADC_ABSOLUTE` for 10V range configuration: 1mV ... 10500mV
- `IO_ADC_ABSOLUTE` 32V inputs: 1mV ... 32780mV (1mV ... 33333mV for TTC32 variants)
- `IO_ADC_RATIOMETRIC`: 1mV ... 5000mV
- `IO_ADC_CURRENT`: 1uA ... 25000uA
- `IO_ADC_RESISTIVE`: 1 Ohm ... 65534 Ohm

The safety configuration is applicable for the following channels:

- `IO_ADC_00 .. IO_ADC_01`
- `IO_ADC_10 .. IO_ADC_15`
- `IO_ADC_20 .. IO_ADC_21`

Field Documentation

ubyte2 _io_adc_safety_conf::adc_val_lower

Lower limit for analog input value

ubyte2 _io_adc_safety_conf::adc_val_upper

Upper limit for analog input value

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

- **IO_ADC.h**



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	
_io_brbl_can_id_ Struct Reference			Data Fields

CAN ID structure. [More...](#)

```
#include <IO_BRBL.h>
```

Data Fields

ubyte4 extended

ubyte4 ID

Detailed Description

CAN ID structure.

Field Documentation

ubyte4 _io_brbl_can_id_::extended

Type of CAN identifier to be used. Valid values are: 0 ... standard CAN identifier is used 1 ... extended CAN identifier is used

ubyte4 _io_brbl_can_id_::ID

The CAN identifier (LSB must start at bit 0): bit 0-10 ... if standard CAN identifier is used bit 0-28 ... if extended CAN identifier is used

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

- **IO_BRBL.h**

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_brbl_can_param Struct Reference

Branding block CAN parameter structure. [More...](#)

```
#include <IO_BRBL.h>
```

Data Fields

ubyte4 CANBaudrate

ubyte4 CANChannel

IO_BRBL_CAN_ID CANDownloadID

IO_BRBL_CAN_ID CANUploadID

IO_BRBL_CAN_ID UDSONCANRxID

IO_BRBL_CAN_ID UDSONCANTxID

IO_BRBL_CAN_ID UDSONCANFuncRxID

Detailed Description

Branding block CAN parameter structure.

Field Documentation

ubyte4 _io_brbl_can_param::CANBaudrate

Baud rate in kbit/s used for CAN communication.

ubyte4 _io_brbl_can_param::CANChannel

The channel used for CAN communication.

IO_BRBL_CAN_ID _io_brbl_can_param::CANDownloadID

The CAN identifier used for download direction (TTC-Downloader -> target).

IO_BRBL_CAN_ID _io_brbl_can_param::CANUploadID

The CAN identifier used for upload direction (target -> TTC-Downloader).

IO_BRBL_CAN_ID _io_brbl_can_param::UDSOnCANFuncRxID

The CAN identifier used for download direction (diagnostic tester -> target).

IO_BRBL_CAN_ID _io_brbl_can_param::UDSONCANRxID

The CAN identifier used for download direction (diagnostic tester -> target).

IO_BRBL_CAN_ID _io_brbl_can_param::UDSONCANTxID

The CAN identifier used for upload direction (target -> diagnostic tester).

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

- **IO_BRBL.h**

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_brbl_dids Struct Reference

Entry definition for DID table. [More...](#)

```
#include <IO_BRBL.h>
```

Data Fields

ubyte2 **Did**

ubyte2 **DidLength**

const **ubyte1** * **DidData**

Detailed Description

Entry definition for DID table.

Field Documentation

ubyte2 _io_brbl_dids::Did

Data Identifier

const ubyte1* _io_brbl_dids::DidData

Pointer to DID data. DID data must be stored MSB first (Array index 0 -> MSB, index X -> LSB) and must not exceed DidLength.

ubyte2 _io_brbl_dids::DidLength

Length of Data Identifier

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

- **IO_BRBL.h**



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_can_data_frame Struct Reference

CAN data frame. [More...](#)

```
#include <IO_CAN.h>
```

Data Fields

ubyte1 data [8]

ubyte1 length

ubyte1 id_format

ubyte4 id

Detailed Description

CAN data frame.

Stores a data frame for the CAN communication.

Field Documentation

ubyte1 _io_can_data_frame::data[8]

data buffer

ubyte4 _io_can_data_frame::id

ID for CAN communication

ubyte1 _io_can_data_frame::id_format

standard or extended format

ubyte1 _io_can_data_frame::length

number of words in transmit buffer

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

- **IO_CAN.h**



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

_io_driver_di_limits Struct Reference

Voltage limits for digital inputs. [More...](#)

```
#include <IO_DIO.h>
```

Detailed Description

Voltage limits for digital inputs.

Contains the thresholds for valid low- and high-levels for digital inputs.

The range for the low-level is defined by the voltages `low_thresh1` and `low_thresh2`, where `low_thresh1` is the lower limit for a low-level and `low_thresh2` the upper limit.

The range for the high-level is defined by the voltages `high_thresh1` and `high_thresh2`, where `high_thresh1` is the lower limit for a high-level and `high_thresh2` the upper limit.

The value of `low_thresh1` must always be smaller than `low_thresh2` and `high_thresh1` must always be smaller than `high_thresh2`.

It is possible to configure a hysteresis by setting `low_thresh2` bigger than `high_thresh1`. In this mode an already detected logic level will be hold as long as the analog voltage varies within the hysteresis band `low_thresh2 - high_thresh1`. Please see the following examples:

Examples:

```
// voltage limits without hysteresis
IO_DRIVER_DI_LIMITS limits1 = { 0, 2000, 3000,
    5000 };
```

In the above example `limits1` defines the range 0-2000mV as valid low-level and 3000-5000mV as valid high-level. The voltage range between 2000 and 3000mV represents an invalid area. In this case the error code `IO_E_DI_INVALID_VOLTAGE` will be returned.

```
// voltage limits with hysteresis
IO_DRIVER_DI_LIMITS limits2 = { 0, 3000, 2000,
    5000 };
```

In the above example `limits1` defines the range 0-2000mV as valid low-level and 3000-5000mV as valid high-level. The range 2000-3000mV forms a hysteresis. If the voltage value rises or drops within this band (between 2000 and 3000mV) the former state of the channel will be held.

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

- [IO_DIO.h](#)



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_driver_rst_info Struct Reference

Reset information. [More...](#)

```
#include <IO_Driver.h>
```

Data Fields

IO_DRIVER_RESET_REASON	reset_reason
------------------------	--------------

ubyte2	reset_counter
--------	---------------

IO_DRIVER_TRAP_INFO	trap_info
---------------------	-----------

Detailed Description

Reset information.

Field Documentation

ubyte2 _io_driver_rst_info::reset_counter

Reset counter, 0 means it was a power-on reset

IO_DRIVER_RESET_REASON _io_driver_rst_info::reset_reason

Reset reason

IO_DRIVER_TRAP_INFO _io_driver_rst_info::trap_info

Trap/Exception information

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

- **IO_Driver.h**



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_driver_safety_conf Struct Reference

Driver Safety Configuration. [More...](#)

```
#include <IO_Driver.h>
```

Data Fields

ubyte1 safety_switch_type

ubyte1 glitch_filter_time

ubyte2 cycle_time

DIAG_ERR_CALLBACK error_callback

Detailed Description

Driver Safety Configuration.

This structure is used to pass the configuration for a safety critical application to the IO-Driver.

Field Documentation

ubyte2 _io_driver_safety_conf::cycle_time

Maximum Cycle time for the IO-Driver task (500..25000us)

DIAG_ERR_CALLBACK

_io_driver_safety_conf::error_callback

Callback function that will be called in case of non-fatal errors.
Can be set to NULL if no function shall be called

ubyte1 _io_driver_safety_conf::glitch_filter_time

Filter time for debouncing errors (10..180ms)

ubyte1 _io_driver_safety_conf::safety_switch_type

Declares whether an internal, external or no safety switch is used (`IO_DRIVER_SAFETY_SWITCH_INT`, `IO_DRIVER_SAFETY_SWITCH_EXT` Or `IO_DRIVER_SAFETY_SWITCH_NONE`)

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

- **IO_Driver.h**



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_driver_trap_info Struct Reference

Contains information regarding traps/exceptions. [More...](#)

```
#include <IO_Driver.h>
```

Data Fields

ubyte2 trap_id

ubyte2 cpu_tfr

ubyte2 cpu_trapstat

ubyte2 cpu_eccstat

ubyte2 cpu_pecon

ubyte4 fault_location

Detailed Description

Contains information regarding traps/exceptions.

Field Documentation

ubyte2 _io_driver_trap_info::cpu_eccstat

Content of CPU register ECCSTAT before reset

ubyte2 _io_driver_trap_info::cpu_pecon

Content of CPU register PECON before reset

ubyte2 _io_driver_trap_info::cpu_tfr

Content of CPU register TFR before reset

ubyte2 _io_driver_trap_info::cpu_trapstat

Content of CPU register TRAPSTAT before reset

ubyte4 _io_driver_trap_info::fault_location

Fault location. In most cases the address given in this variable points to the location following the one which caused the exception/trap. This fault location is only a hint where to search. It does not point to the exact location of the problem. The fault location is only available for system stack over-/under-flow and class B traps.

ubyte2 _io_driver_trap_info::trap_id

Trap-id. Can be one of:

- 0: no trap occurred
- 1: Service Request 0 trap occurred
- 2: System stack overflow trap occurred
- 3: System stack underflow trap occurred
- 4: Software trap occurred
- 5: Class B trap occurred

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

- **IO_Driver.h**

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_pid_config **Struct Reference**

PID configuration structure. [More...](#)

```
#include <IO_PID.h>
```

Data Fields

sbyte4 Kff

sbyte4 Kp

sbyte4 Ki

sbyte4 Kd

sbyte4 max_limit

sbyte4 min_limit

Detailed Description

PID configuration structure.

Data structure that contains all configuration parameters for PID control.

The following formula is used for the PID:

```
output = Kff/10^3 * setpoint + Kp/10^3 * error  
        + Kd/10^4 * delta(error) + Ki/10^4 *  
        sum(error)
```

Please note that Kff and Kp are scaled by a factor of 1,000 and Ki and Kd by a factor of 10,000 to improve the resolution on small gains.

Remarks

- Typical values for Kff are between 10.000 and 15.000
- Typical values for Ki are around 20.000
- Typical values for Kp and Kd are smaller than 1.000

Attention

Please keep in mind that the unit of `output` has to be

- Volts for voltage outputs
- Duty cycle in digits [0 .. 65535] for PWM outputs

Field Documentation

sbyte4 _io_pid_config::Kd

Derivative gain constant scaled by a factor of 10,000. A value of 0 disables the derivative term in the PID.

sbyte4 _io_pid_config::Kff

Gain constant for feed-forward control scaled by a factor of 1,000. A value of 0 disables the feed-forward control, a value of 1,000 leads to a 1:1 loop through of the setpoint.

sbyte4 _io_pid_config::Ki

Integral gain constant scaled by a factor of 10,000. A value of 0 disables the integral term in the PID.

sbyte4 _io_pid_config::Kp

Proportional gain constant scaled by a factor of 1,000. A value of 0 disables the proportional term in the PID.

sbyte4 _io_pid_config::max_limit

Upper boundary for output value. The output will saturate on this value.

sbyte4 _io_pid_config::min_limit

Lower boundary for output value. The output will saturate on this value.

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

- **IO_PID.h**



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_pwd_cplx_safety _conf Struct Reference

Safety configuration for the Complex PWD inputs. [More...](#)

```
#include <IO_PWD.h>
```

Data Fields

ubyte2 pwd_freq_val_upper

ubyte2 pwd_freq_val_lower

ubyte4 pwd_pulse_val_upper

ubyte4 pwd_pulse_val_lower

Detailed Description

Safety configuration for the Complex PWD inputs.

Stores all relevant safety configuration parameters for the Complex PWD inputs.

Field Documentation

ubyte2 _io_pwd_cplx_safety_conf::pwd_freq_val_lower

Lower PWD frequency limit in Hz [1Hz..65534Hz]

ubyte2 _io_pwd_cplx_safety_conf::pwd_freq_val_upper

Upper PWD frequency limit in Hz [1Hz..65534Hz]

ubyte4 _io_pwd_cplx_safety_conf::pwd_pulse_val_lower

Lower limit of the pulse time in us for a single PWD pulse [1us..4294967294us].

Depending on the configuration of the channel (whether the high or low time of the pulse should be measured) this value represents the respective time segment.

ubyte4 _io_pwd_cplx_safety_conf::pwd_pulse_val_upper

Upper limit of the pulse time in us for a single PWD pulse [1us..4294967294us].

Depending on the configuration of the channel (whether the high or low time of the pulse should be measured) this value represents the respective time segment.

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

- [IO_PWD.h](#)

Generated on Mon Nov 16 2020 16:59:48 for HY-TTC 30 Family C API Manual by

 1.8.2

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_pwd_inc_safety_ conf Struct Reference

Safety configuration for the Incremental or Counter PWD inputs.
[More...](#)

```
#include <IO_PWD.h>
```

Data Fields

```
ubyte2  pwd_cnt_val_upper
```

```
ubyte2  pwd_cnt_val_lower
```

Detailed Description

Safety configuration for the Incremental or Counter PWD inputs.

Stores all relevant safety configuration parameters for the Incremental PWD inputs.

Field Documentation

ubyte2 _io_pwd_inc_safety_conf::pwd_cnt_val_lower

Lower PWD counter limit [1..65534]

ubyte2 _io_pwd_inc_safety_conf::pwd_cnt_val_upper

Upper PWD counter limit [1..65534]

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

- **IO_PWD.h**

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

[Data Fields](#)

_io_pwd_pulse_samples **Struct Reference**

PWD pulse-width data structure. [More...](#)

```
#include <IO_PWD.h>
```

Data Fields

ubyte1 **pulse_samples_count**

ubyte4 **pulse_sample** [IO_PWD_MAX_PULSE_SAMPLES]

Detailed Description

PWD pulse-width data structure.

stores each captured pulse-width for one measurement.

Field Documentation

ubyte4

_io_pwd_pulse_samples::pulse_sample[IO_PWD_MAX_PULSE_SAMPLES]

stores each captured pulse-width for one measurement

ubyte1 _io_pwd_pulse_samples::pulse_samples_count

number of pulse_samples

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

- **[IO_PWD.h](#)**

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_pwm_current_queue Struct Reference

PWM current measurement queue. [More...](#)

```
#include <IO_PWM.h>
```

Data Fields

ubyte1 count

bool overrun

ubyte2 values [IO_PWM_CURRENT_QUEUE_MAX]

Detailed Description

PWM current measurement queue.

Stores results of the equidistant current measurement.

The queue holds all current measurement since the last retrieval via the step function `IO_PWM_GetCur()`.

Field Documentation

ubyte1 _io_pwm_current_queue::count

Number of results stored in the queue

bool _io_pwm_current_queue::overrun

Signal queue overrun. TRUE means queue is full and older measurement results may have been dropped. FALSE means queue is not overrun.

ubyte2 _io_pwm_current_queue::values[IO_PWM_CURRENT_QUEUE_MAX]

Buffer holding the measurement values values

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

- **IO_PWM.h**



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_pwm_current_safety_conf Struct Reference

Safety configuration for the PWM outputs. [More...](#)

```
#include <IO_PWM.h>
```

Data Fields

ubyte2 **current_tolerance**

ubyte4 **dead_time**

IO_PWM_SAFETY_CONF **pwm_safety_conf**

Detailed Description

Safety configuration for the PWM outputs.

Stores all relevant safety configuration parameters for the PWM outputs.

Attention

For the current check to work properly it is necessary that the application does not change the set-point of the PID controller before the dead time elapses! for example if a dead time of 10ms is set, the application shall change the set-point for the current value less frequently than 10ms!

To disable the current check, set the value of the parameters `current_tolerance` to 65535 and `dead_time` to 4294967295.

Field Documentation

ubyte2 _io_pwm_current_safety_conf::current_tolerance

Tolerance of the measured electric current in Milli-Ampere.
Allowed values: 50mA .. 500mA

ubyte4 _io_pwm_current_safety_conf::dead_time

Time until current should have reached its target value in
Microseconds. Allowed values: 5000us ..10000000us

IO_PWM_SAFETY_CONF _io_pwm_current_safety_conf::pwm_safety_conf

Safety configuration for PWM signal

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

- **IO_PWM.h**



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

_io_pwm_safety_conf Struct Reference

Safety configuration for the PWM outputs. [More...](#)

```
#include <IO_PWM.h>
```

Data Fields

IO_PIN safety_switch

ubyte2 margin_lower_lim

ubyte2 margin_upper_lim

ubyte2 duty_cycle_tolerance

Detailed Description

Safety configuration for the PWM outputs.

Stores all relevant safety configuration parameters for the PWM outputs.

Field Documentation

ubyte2 _io_pwm_safety_conf::duty_cycle_tolerance

Tolerance in microseconds for the set duty cycle. This limit can be adjusted within 100us to 200us

ubyte2 _io_pwm_safety_conf::margin_lower_lim

Limit in microseconds for the lower minimum pulse. This limit can be adjusted within 50us to 100us.

ubyte2 _io_pwm_safety_conf::margin_upper_lim

Limit in microseconds for the upper minimum pulse pause. This limit can be adjusted within 50us to 150us

IO_PIN _io_pwm_safety_conf::safety_switch

Connected safety switch ([IO_SAFETY_SWITCH_0](#) or [IO_SAFETY_SWITCH_1](#), or alternatively [IO_SAFETY_SWITCH_NONE](#) if neither safety switch is connected).

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

- [IO_PWM.h](#)

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Fields

ApdbType Struct Reference

APDB structure. [More...](#)

```
#include <Apdb.h>
```

Data Fields

uint32 **ApdbVersion**

uint32 **FlashDate**

uint32 **BuildDate**

uint32 **NodeType**

uint32 **CrcStartAddress**

uint32 **CodeSize**

uint32 **LegacyApplicationCrc**

uint32 **ApplicationCrc**

uint32	NodeNumber
uint32	CrcSeed
uint32	Flags
uint32	Hook1
uint32	Hook2
uint32	Hook3
uint32	MainAddress
CanIdType	CanDownloadId
CanIdType	CanUploadId
uint32	LegacyHeaderCrc
uint32	ApplicationVersion
uint32	CanBaudrate
uint32	CanChannel
uint32	Password
uint32	MagicSeed
uint8	TargetIpAddress [4]
uint8	SubnetMask [4]

uint8	DIMulticastIpAddress [4]
-------	---------------------------------

uint32	DebugKey
--------	-----------------

uint32	AbrdTimeout
--------	--------------------

uint8	ManufacturerId
-------	-----------------------

uint8	ApplicationId
-------	----------------------

uint16	Reserved
--------	-----------------

uint32	HeaderCrc
--------	------------------

Detailed Description

APDB structure.

Data structure for accessing the Application Descriptor Block.

Field Documentation

uint32 ApdbType::AbrdTimeout

The timeout for automatic CAN baud rate detection.

uint32 ApdbType::ApdbVersion

The APDB version: bit 0-7 ... minor number bit 8-15 ... major number

uint32 ApdbType::ApplicationCrc

CRC-32 value calculated over the application or if a CRC table is used, CRC-32 value calculated over the CRC table (automatically provided by the TTC-Downloader).

uint8 ApdbType::ApplicationId

The application identifier (must be provided by the customer).

uint32 ApdbType::ApplicationVersion

The application version (must be provided by the customer): bit 0-15 ... revision number bit 16-23 ... minor number bit 24-31 ... major number

uint32 ApdbType::BuildDate

The application's build date (must be provided by the customer).

uint32 ApdbType::CanBaudrate

Baud rate in kbit/s used for CAN communication.

uint32 ApdbType::CanChannel

The channel used for CAN communication.

CanIdType ApdbType::CanDownloadId

The CAN identifier used for download direction (TTC-Downloader -> target).

CanIdType ApdbType::CanUploadId

The CAN identifier used for upload direction (target -> TTC-Downloader).

uint32 ApdbType::CodeSize

Code size in bytes (used for CRC calculation) or if a CRC table is used, number of CRC table entries (automatically provided by the TTC-Downloader).

uint32 ApdbType::CrcSeed

Seed for application CRC calculation (automatically provided by the TTC-Downloader).

uint32 ApdbType::CrcStartAddress

Start address for CRC calculation or if a CRC table is used, start address of the CRC table (automatically provided by the TTC-Downloader).

uint32 ApdbType::DebugKey

Debug key for booting the device in debug mode.

uint8 ApdbType::DIMulticastIpAddress[4]

Multicast IP address of the TTC-Downloader (most significant byte first).

uint32 ApdbType::Flags

Predefined application flags can be specified here. Following flags can be specified: bit 0 ... Auto-BaudrateDetectionEnable 0 = disable automatic baud rate detection 1 = enable automatic baud rate detection

uint32 ApdbType::FlashDate

The date when the application has been flashed (automatically provided by the TTC-Downloader).

uint32 ApdbType::HeaderCrc

The CRC value calculated over the whole APDB (except HeaderCRC field).

uint32 ApdbType::Hook1

Custom hook 1.

uint32 ApdbType::Hook2

Custom hook 2.

uint32 ApdbType::Hook3

Custom hook 3.

uint32 ApdbType::LegacyApplicationCrc

Legacy application CRC for flash checker (automatically provided by the TTC-Downloader).

uint32 ApdbType::LegacyHeaderCrc

Legacy header CRC for flash checker (automatically provided by the TTC-Downloader).

uint32 ApdbType::MagicSeed

Seed for CRC calculation with the MCHK HW module.

uint32 ApdbType::MainAddress

The application's vector address. Note that the bootloader uses this address to start the application after reset/power-up.

uint8 ApdbType::ManufacturerId

The manufacturer identifier (must be provided by the customer). The generic ID 0xFF shall be used for private use.

uint32 ApdbType::NodeNumber

The unique node number used to identify nodes of a CODESYS application. Note that only values from 0 to 127 are allowed.

uint32 ApdbType::NodeType

The hardware type the application is built for.

uint32 ApdbType::Password

The password hash for memory access. Set this field to 0 or 0xFFFFFFFF to disable password protection. Note that it is highly recommended to set a password upon application download with the TTC-Downloader.

uint16 ApdbType::Reserved

Reserved for future use. Shall be set to 0.

uint8 ApdbType::SubnetMask[4]

Subnet mask for Ethernet download (most significant byte first).

uint8 ApdbType::TargetIpAddress[4]

Target IP address for Ethernet download (most significant byte first).

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

- [Apdb.h](#)



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	
can_id Struct Reference			

Data Fields

CAN ID structure. [More...](#)

```
#include <TypesGen.h>
```

Data Fields

uint32	Extended
uint32	Id

Detailed Description

CAN ID structure.

Field Documentation

uint32 can_id::Extended

Type of CAN identifier to be used. Valid values are: 0 ... standard CAN identifier is used 1 ... extended CAN identifier is used

uint32 can_id::Id

The CAN identifier (LSB must start at bit 0): bit 0-10 ... if standard CAN identifier is used bit 0-28 ... if extended CAN identifier is used

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

- **TypesGen.h**

Main Page	Related Pages	Data Structures	Files
Data Structures	Data Structure Index	Data Fields	

Data Structure Index

A C _				
A	_	<code>_io_brbl_can_param</code>	<code>_io_driver_safety_conf</code>	<code>_io_p</code>
		<code>_io_brbl_dids</code>	<code>_io_driver_trap_info</code>	<code>_io_p</code>
ApdbType	<code>_diag_errorcode</code>	<code>_io_can_data_frame</code>	<code>_io_pid_config</code>	<code>_io_pwn</code>
C	<code>_io_adc_safety_conf</code>	<code>_io_driver_di_limits</code>	<code>_io_pwd_cplx_safety_conf</code>	<code>_io_</code>
	<code>_io_brbl_can_id_</code>	<code>_io_driver_rst_info</code>	<code>_io_pwd_inc_safety_conf</code>	
can_id				
A C _				



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page				Related Pages				Data Structures				Files								
Data Structures				Data Structure Index				Data Fields												
All		Variables																		
a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	r	s	t	u	v	

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- a -

- AbridTimeout : [ApdbType](#)
- adc_val_lower : [_io_adc_safety_conf](#)
- adc_val_upper : [_io_adc_safety_conf](#)
- ApdbVersion : [ApdbType](#)
- ApplicationCrc : [ApdbType](#)
- ApplicationId : [ApdbType](#)
- ApplicationVersion : [ApdbType](#)

- b -

- BuildDate : [ApdbType](#)

- c -

- CANBaudrate : [_io_brbl_can_param](#)
- CanBaudrate : [ApdbType](#)
- CanChannel : [ApdbType](#)
- CANChannel : [_io_brbl_can_param](#)

- CanDownloadId : **ApdbType**
- CANDownloadID : **_io_brbl_can_param**
- CANUploadID : **_io_brbl_can_param**
- CanUploadId : **ApdbType**
- CodeSize : **ApdbType**
- count : **_io_pwm_current_queue**
- cpu_eccstat : **_io_driver_trap_info**
- cpu_pecon : **_io_driver_trap_info**
- cpu_tfr : **_io_driver_trap_info**
- cpu_trapstat : **_io_driver_trap_info**
- CrcSeed : **ApdbType**
- CrcStartAddress : **ApdbType**
- current_tolerance : **_io_pwm_current_safety_conf**
- cycle_time : **_io_driver_safety_conf**

- d -

- data : **_io_can_data_frame**
- dead_time : **_io_pwm_current_safety_conf**
- DebugKey : **ApdbType**
- device_num : **_diag_errorcode**
- Did : **_io_brbl_dids**
- DidData : **_io_brbl_dids**
- DidLength : **_io_brbl_dids**
- DIMulticastIpAddress : **ApdbType**
- duty_cycle_tolerance : **_io_pwm_safety_conf**

- e -

- error_callback : **_io_driver_safety_conf**
- error_code : **_diag_errorcode**
- extended : **_io_brbl_can_id_**
- Extended : **can_id**

- f -

- fault_location : `_io_driver_trap_info`
- faulty_value : `_diag_errorcode`
- Flags : `ApdbType`
- FlashDate : `ApdbType`

- g -

- glitch_filter_time : `_io_driver_safety_conf`

- h -

- HeaderCrc : `ApdbType`
- Hook1 : `ApdbType`
- Hook2 : `ApdbType`
- Hook3 : `ApdbType`

- i -

- Id : `can_id`
- id : `_io_can_data_frame`
- ID : `_io_brbl_can_id_`
- id_format : `_io_can_data_frame`

- k -

- Kd : `_io_pid_config`
- Kff : `_io_pid_config`
- Ki : `_io_pid_config`
- Kp : `_io_pid_config`

- l -

- LegacyApplicationCrc : `ApdbType`
- LegacyHeaderCrc : `ApdbType`
- length : `_io_can_data_frame`

- m -

- MagicSeed : **ApdbType**
- MainAddress : **ApdbType**
- ManufacturerId : **ApdbType**
- margin_lower_lim : **_io_pwm_safety_conf**
- margin_upper_lim : **_io_pwm_safety_conf**
- max_limit : **_io_pid_config**
- min_limit : **_io_pid_config**

- n -

- NodeNumber : **ApdbType**
- NodeType : **ApdbType**

- o -

- overrun : **_io_pwm_current_queue**

- p -

- Password : **ApdbType**
- pulse_sample : **_io_pwd_pulse_samples**
- pulse_samples_count : **_io_pwd_pulse_samples**
- pwd_cnt_val_lower : **_io_pwd_inc_safety_conf**
- pwd_cnt_val_upper : **_io_pwd_inc_safety_conf**
- pwd_freq_val_lower : **_io_pwd_cplx_safety_conf**
- pwd_freq_val_upper : **_io_pwd_cplx_safety_conf**
- pwd_pulse_val_lower : **_io_pwd_cplx_safety_conf**
- pwd_pulse_val_upper : **_io_pwd_cplx_safety_conf**
- pwm_safety_conf : **_io_pwm_current_safety_conf**

- r -

- Reserved : **ApdbType**

- reset_counter : [_io_driver_rst_info](#)
- reset_reason : [_io_driver_rst_info](#)

- S -

- safety_switch : [_io_pwm_safety_conf](#)
- safety_switch_type : [_io_driver_safety_conf](#)
- SubnetMask : [ApdbType](#)

- t -

- TargetIpAddress : [ApdbType](#)
- trap_id : [_io_driver_trap_info](#)
- trap_info : [_io_driver_rst_info](#)

- u -

- UDSONCANFuncRxID : [_io_brbl_can_param](#)
- UDSONCANRxID : [_io_brbl_can_param](#)
- UDSONCANTxID : [_io_brbl_can_param](#)

- v -

- values : [_io_pwm_current_queue](#)

Main Page			Related Pages			Data Structures			Files											
Data Structures			Data Structure Index			Data Fields														
All		Variables																		
a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	r	s	t	u	v	

- a -

- AbridTimeout : **ApdbType**
- adc_val_lower : **_io_adc_safety_conf**
- adc_val_upper : **_io_adc_safety_conf**
- ApdbVersion : **ApdbType**
- ApplicationCrc : **ApdbType**
- ApplicationId : **ApdbType**
- ApplicationVersion : **ApdbType**

- b -

- BuildDate : **ApdbType**

- c -

- CANBaudrate : **_io_brbl_can_param**
- CanBaudrate : **ApdbType**
- CanChannel : **ApdbType**
- CANChannel : **_io_brbl_can_param**
- CanDownloadId : **ApdbType**

- CANDownloadID : `_io_brbl_can_param`
- CANUploadID : `_io_brbl_can_param`
- CanUploadId : `ApdbType`
- CodeSize : `ApdbType`
- count : `_io_pwm_current_queue`
- cpu_eccstat : `_io_driver_trap_info`
- cpu_pecon : `_io_driver_trap_info`
- cpu_tfr : `_io_driver_trap_info`
- cpu_trapstat : `_io_driver_trap_info`
- CrcSeed : `ApdbType`
- CrcStartAddress : `ApdbType`
- current_tolerance : `_io_pwm_current_safety_conf`
- cycle_time : `_io_driver_safety_conf`

- d -

- data : `_io_can_data_frame`
- dead_time : `_io_pwm_current_safety_conf`
- DebugKey : `ApdbType`
- device_num : `_diag_errorcode`
- Did : `_io_brbl_dids`
- DidData : `_io_brbl_dids`
- DidLength : `_io_brbl_dids`
- DIMulticastIpAddress : `ApdbType`
- duty_cycle_tolerance : `_io_pwm_safety_conf`

- e -

- error_callback : `_io_driver_safety_conf`
- error_code : `_diag_errorcode`
- extended : `_io_brbl_can_id_`
- Extended : `can_id`

- f -

- fault_location : `_io_driver_trap_info`

- faulty_value : **_diag_errorcode**
- Flags : **ApdbType**
- FlashDate : **ApdbType**

- g -

- glitch_filter_time : **_io_driver_safety_conf**

- h -

- HeaderCrc : **ApdbType**
- Hook1 : **ApdbType**
- Hook2 : **ApdbType**
- Hook3 : **ApdbType**

- i -

- Id : **can_id**
- id : **_io_can_data_frame**
- ID : **_io_brbl_can_id_**
- id_format : **_io_can_data_frame**

- k -

- Kd : **_io_pid_config**
- Kff : **_io_pid_config**
- Ki : **_io_pid_config**
- Kp : **_io_pid_config**

- l -

- LegacyApplicationCrc : **ApdbType**
- LegacyHeaderCrc : **ApdbType**
- length : **_io_can_data_frame**

- m -

- MagicSeed : **ApdbType**
- MainAddress : **ApdbType**
- ManufacturerId : **ApdbType**
- margin_lower_lim : **_io_pwm_safety_conf**
- margin_upper_lim : **_io_pwm_safety_conf**
- max_limit : **_io_pid_config**
- min_limit : **_io_pid_config**

- n -

- NodeNumber : **ApdbType**
- NodeType : **ApdbType**

- o -

- overrun : **_io_pwm_current_queue**

- p -

- Password : **ApdbType**
- pulse_sample : **_io_pwd_pulse_samples**
- pulse_samples_count : **_io_pwd_pulse_samples**
- pwd_cnt_val_lower : **_io_pwd_inc_safety_conf**
- pwd_cnt_val_upper : **_io_pwd_inc_safety_conf**
- pwd_freq_val_lower : **_io_pwd_cplx_safety_conf**
- pwd_freq_val_upper : **_io_pwd_cplx_safety_conf**
- pwd_pulse_val_lower : **_io_pwd_cplx_safety_conf**
- pwd_pulse_val_upper : **_io_pwd_cplx_safety_conf**
- pwm_safety_conf : **_io_pwm_current_safety_conf**

- r -

- Reserved : **ApdbType**

- reset_counter : [_io_driver_rst_info](#)
- reset_reason : [_io_driver_rst_info](#)

- S -

- safety_switch : [_io_pwm_safety_conf](#)
- safety_switch_type : [_io_driver_safety_conf](#)
- SubnetMask : [ApdbType](#)

- t -

- TargetIpAddress : [ApdbType](#)
- trap_id : [_io_driver_trap_info](#)
- trap_info : [_io_driver_rst_info](#)

- u -

- UDSONCANFuncRxID : [_io_brbl_can_param](#)
- UDSONCANRxID : [_io_brbl_can_param](#)
- UDSONCANTxID : [_io_brbl_can_param](#)

- v -

- values : [_io_pwm_current_queue](#)

















HY-TTC 30 Family C















API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
File List	Globals		

File List

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

 Apdb.h	APDB define for bootloader
 ApdbCfg.h	Definitions for Apdb.h
 DIAG_Constants.h	Global defines for IO Driver diagnostic state machine and WD
 DIAG_Functions.h	Auxiliary functions for the diagnostic state machine
 IO_ADC.h	IO Driver functions for ADC
 IO_BRBL.h	API for accessing data in the branding block of the ECU
 IO_CAN.h	IO Driver functions for CAN communication
 IO_Constants.h	Global defines for IO Driver
 IO_Crypt.h	API for I/O driver cryptographic functions
 IO_DIO.h	IO Driver functions for Digital Input/Output
 IO_Driver.h	High level interface to IO Driver
 IO_EEPROM.h	IO Driver functions for EEPROM
 IO_EEPROM_Preload.h	Pre-load functions for EEPROM
 IO_LED.h	IO driver functions for LED

 IO_NodeID.h	IO Driver functions for reading the NodeID pins
 IO_PID.h	Contains the data structure for configuring the PID controller
 IO_Pins.h	Global IO Pin defines for IO Driver
 IO_POWER.h	IO Driver functions for Power control
 IO_PVG.h	IO Driver functions for PVG channels
 IO_PWD.h	IO Driver functions for timer input channels
 IO_PWM.h	IO Driver functions for PWM channels
 IO_RTC.h	RTC functions, provides exact timing functions
 IO_UART.h	IO Driver functions for UART communication
 IO_Vout.h	IO Driver functions for voltage outputs
 IO_WD.h	IO-Driver for the Window Watchdog
 IO_WDTimer.h	IO Driver functions for the CPU's Watchdog timer
 ptypes_xe167.h	Primitive data types
 TypesGen.h	Types header file



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
File List	Globals		
External	LogisticTypes	Apdb	

Data Structures | Macros

Apdb.h File Reference

APDB define for bootloader. [More...](#)

```
#include "TypesGen.h"
```

Data Structures

```
struct ApdbType  
    APDB structure. More...
```

Macros

```
#define APDB_VERSION (0x00000206lu)
```

```
#define APDB_SIZE (0x80u)
```

APDB flags

Defined APDB flags.

```
#define APDB_FLAGS_ABRD_ENABLE (0x00000001lu)
```

```
#define APDB_FLAGS_CRC64_ENABLE (0x40000000lu)
```

```
#define APDB_FLAGS_MULTI_APP (0x80000000lu)
```

Detailed Description

APDB define for bootloader.

Contains the definition for the application database. This database is used by the bootloader.

The bootloader needs this information to determine where the application actually starts. For this reason the field "MainAddress" must be provided by the application.

APDB Usage:

- **Example for APDB definition**

APDB Code Example

Example for using the APDB

Example for APDB definition in an application

```
volatile const ApdbType Apdb =
{
    APDB_VERSION,          // APDB version
    {0},                   // Flash date
                           // (provided by TTC-Downloader)
                           // Build date
    { (((RTS_TTC_FLASH_DATE_YEAR)    & 0x0FFF)
      << 0) |
      (((RTS_TTC_FLASH_DATE_MONTH)  & 0x0F )
      << 12) |
      (((RTS_TTC_FLASH_DATE_DAY)    & 0x1F )
      << 16) |
      (((RTS_TTC_FLASH_DATE_HOUR)   & 0x1F )
      << 21) |
      (((RTS_TTC_FLASH_DATE_MINUTE) & 0x3F )
      << 26)) },
    0,                     // Node type
    0,                     // CRC start
                           // address (provided by TTC-Downloader)
    0,                     // Code size
                           // (provided by TTC-Downloader)
    0,                     // Legacy
                           // application CRC (provided by TTC-
                           // Downloader)
    0,                     // Application CRC
                           // (provided by TTC-Downloader)
    1,                     // Node number
    0,                     // CRC seed
                           // (provided by TTC-Downloader)
    0,                     // Flags
    0,                     // Hook 1
    0,                     // Hook 2
}
```

```

0,                                // Hook 3
APPL_START,                       // Main address,
i.e., application entry point
{0, 1},                           // CAN download ID
(standard format, ID 0x1)
{0, 2},                           // CAN upload ID
(standard format, ID 0x2)
0,                                // Legacy header
CRC (provided by TTC-Downloader)
                                // Application
version (major.minor.revision)
(((uint32)REVISION_NUMBER) << 0) |
((uint32)    MINOR_NUMBER) << 16) |
((uint32)    MAJOR_NUMBER) << 24)),
500,                              // CAN baud rate in
kbps
0,                                // CAN channel
0,                                // Password
(disable password protection)
0,                                // Magic seed
{ 10, 100, 30, 200},             // Target IP
address
{255, 255, 0, 0},                // Subnet mask
{239, 0, 0, 1},                  // Multicast IP
address
0,                                // Debug key
0,                                // Automatic baud
rate detection timeout
0x00,                             // Manufacturer ID
(The generic ID 0xFF shall be used for
private use.)
0x00,                             // Application ID
{0},                              // Reserved, must
be set to zero
0                                 // Header CRC
(provided by TTC-Downloader)

```

```
} ;
```

Copyright (c) TTControl. All rights reserved. Confidential and proprietary.

Macro Definition Documentation

#define APDB_FLAGS_ABRD_ENABLE (0x00000001lu)

Enables automatic baudrate detection at start-up (HY-TTC30X family only). Access mode: read/write.

#define APDB_FLAGS_CRC64_ENABLE (0x40000000lu)

Indicates whether or not CRC-64 is used for application CRC. Access mode: read only.

#define APDB_FLAGS_MULTI_APP (0x80000000lu)

Indicates whether or not the application is distributed over multiple (incoherent) application regions. Access mode: read only.

#define APDB_SIZE (0x80u)

Size of the APDB in bytes

#define APDB_VERSION (0x00000206lu)

Current APDB version is version 2.6.

Generated on Mon Nov 16 2020 16:59:46 for HY-TTC 30 Family C API Manual by

doxygen 1.8.2



HY-TTC 30 Family C

API Manual D-TTC-X-G-

20-001

Main Page		Related Pages		Data Structures		Files
File List		Globals				
inc >						

ApdbCfg.h File Reference

Definitions for [Apdb.h](#). [More...](#)

```
#include "ptypes_xe167.h"
```

Detailed Description

Definitions for **Apdb.h**.

This file defines important settings for **Apdb.h**

Generated on Mon Nov 16 2020 16:59:46 for HY-TTC 30 Family C API Manual by
 1.8.2

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

Data Structures | Typedefs

DIAG_Constants.h File Reference

Global defines for IO Driver diagnostic state machine and WD. [More...](#)

```
#include "IO_Pins.h" #include "ptypes_xe167.h"
```

Data Structures

struct **_diag_errorcode**
Diagnostic Error code structure. [More...](#)

Macros

States of the diagnostic state machine

State information returned by the function DIAG_Status

```
#define DIAG_STATE_INIT 0x03
```

```
#define DIAG_STATE_STARTUP 0x05
```

```
#define DIAG_STATE_MAIN 0x06
```

```
#define DIAG_STATE_SAFE_STATE 0x09
```

```
#define DIAG_STATE_DISABLED 0x00
```

error callback reaction

allowed return values of error callback function

```
#define DIAG_ERR_NOACTION 0x1
```

```
#define DIAG_ERR_SAFESTATE 0x9
```

Typedefs

```
typedef struct _diag_errorcode DIAG_ERRORCODE  
Diagnostic Error code structure.
```

```
typedef ubyte1(* DIAG_ERR_CALLBACK)(ubyte1  
diagnostic_state, DIAG_ERRORCODE  
*error_code)  
Error callback to be used by application  
software.
```

Diagnostic State Machine Error Values

Errors codes that the function `DIAG_Status` returns in parameter `diag_error.error_code`. They are also used for the error callback function.

Remarks

For a detailed description see [Listing of diagnostic errors](#)

```
enum _diag_errortype {  
    DIAG_E_NOERROR = 0,  
    DIAG_E_ADC_LIMITS = 1,  
    DIAG_E_ADC_5V2_SUPPLY = 2,  
    DIAG_E_ADC_SENSOR_SUPPLY = 3,  
    DIAG_E_ADC_KL30_MAIN = 4,  
    DIAG_E_ADC_KL30_CPU = 5,  
    DIAG_E_OVER_TEMPERATURE = 6,  
    DIAG_E_MEM_USER_STACK = 7,  
    DIAG_E_MEM_REGISTER = 8,  
    DIAG_E_MEM_DSRAM = 9,  
};
```

DIAG_E_MEM_PSRAM = 10,
DIAG_E_MEM_DPRAM = 11,
DIAG_E_MEM_ZeroFlag = 12,
DIAG_E_MEM_CarryFlag = 13,
DIAG_E_MEM_NegativeFlag = 14,
DIAG_E_MEM_OverflowFlag = 15,
DIAG_E_MEM_SYS_STACK_OF = 16,
DIAG_E_MEM_SYS_STACK_UF = 17,
DIAG_E_MEM_SR0_TRAP = 18,
DIAG_E_MEM_CLASS_B_TRAP = 19,
DIAG_E_PWM_CURRENT_ZERO = 20,
DIAG_E_PWM_CURRENT_OFFSET =
21,
DIAG_E_PWM_LIMITS_RANGE = 22,
DIAG_E_PWM_LIMITS_TOL = 23,
DIAG_E_PWM_PERIOD_MISMATCH =
24,
DIAG_E_PWM_CURRENT = 25,
DIAG_E_PWM_CURRENT_DEAD_TIME
= 26,
DIAG_E_PWM_CURRENT_OFFS_DRIFT
= 27,
DIAG_E_PWD_LIMITS_FREQ = 28,
DIAG_E_PWD_LIMITS_PULSE_WIDTH
= 29,
DIAG_E_CYCLE_TIME = 30,
DIAG_E_RPP = 31,
DIAG_E_EXT_WD = 32,
DIAG_E_LS_PROT = 33,
DIAG_E_OVD_STARTUP = 34,
DIAG_E_OVD = 35,
DIAG_E_SAFETY_SW_INT = 36,
DIAG_E_SAFETY_SW_EXT = 37,
DIAG_E_SAFETY_SW_SHUT_OFF = 38,
DIAG_E_INVALID_DIAG_STATE = 39,
DIAG_E_INVALID_STARTUP_STATE =
40,
DIAG_E_INVALID_MAIN_STATE = 41,
DIAG_E_WD_STARTUP = 42,
DIAG_E_SR_LowNibble = 43,

```
DIAG_E_SR_HighNibble = 44,  
DIAG_E_FREQ_STARTUP = 45,  
DIAG_E_TIMEOUT = 46,  
DIAG_E_APPL_SAFE_STATE = 47,  
DIAG_E_PLL_VCO_NOT_LOCKED = 48,  
DIAG_E_SW_INTERNAL = 49,  
DIAG_E_INIT_ERROR = 50,  
DIAG_E_INT_WATCHDOG = 51,  
DIAG_E_MEM_SOFTBREAK_TRAP =  
52  
}
```

```
typedef enum _diag_errortype DIAG_ErrorType
```

Detailed Description

Global defines for IO Driver diagnostic state machine and WD.

This header file defines the Error Codes for diagnostic state machine and WD.

Macro Definition Documentation

#define DIAG_ERR_NOACTION 0x1

take no action (ignore the error)

#define DIAG_ERR_SAFESTATE 0x9

enter the safe state (switch off all outputs)

#define DIAG_STATE_DISABLED 0x00

Diagnostic state machine is disabled. This means the IO-Driver has been configured NON-SAFETY

#define DIAG_STATE_INIT 0x03

Diagnostic state machine is in its initial state (safety-outputs not operational)

#define DIAG_STATE_MAIN 0x06

Diagnostic state machine is in main state performing runtime tests (safety-outputs operational)

#define DIAG_STATE_SAFE_STATE 0x09

Diagnostic state machine is in safe state due to errors that have been detected (safety-outputs not operational)

```
#define DIAG_STATE_STARTUP 0x05
```

Diagnostic state machine is in startup state performing startup tests
(safety-outputs not operational)

Typedef Documentation

typedef struct _diag_errorcode DIAG_ERRORCODE

Diagnostic Error code structure.

Stores all relevant error parameters returned from the diagnostic state machine or returned from the WD

typedef enum _diag_errortype DIAG_ErrorType

For a detailed description see [Listing of diagnostic errors](#)

Enumeration Type Documentation

enum _diag_errortype

For a detailed description see [Listing of diagnostic errors](#)

Enumerator:

<i>DIAG_E_NOERROR</i>	no error
<i>DIAG_E_ADC_LIMITS</i>	an ADC limit was exceeded or underrun
<i>DIAG_E_ADC_5V2_SUPPLY</i>	the 5.2V supply voltage deviates from its normal level
<i>DIAG_E_ADC_SENSOR_SUPPLY</i>	the sensor supply voltage deviates from its normal level
<i>DIAG_E_ADC_KL30_MAIN</i>	the KL30 voltage level exceeds or underrun its allowed limits
<i>DIAG_E_ADC_KL30_CPU</i>	the KL30_CPU voltage level exceeds or underrun its allowed limits
<i>DIAG_E_OVER_TEMPERATURE</i>	the board temperature exceeds its allowed

range

DIAG_E_MEM_USER_STACK

a memory block of the user stack is corrupted

DIAG_E_MEM_REGISTER

internal register values are corrupted

DIAG_E_MEM_DSRAM

a memory block of the data S-RAM is corrupted

DIAG_E_MEM_PSRAM

a memory block of the program S-RAM is corrupted

DIAG_E_MEM_DPRAM

a memory block of the dual-port RAM is corrupted

DIAG_E_MEM_ZeroFlag

the zero flag was mistakenly raised during a math operation

DIAG_E_MEM_CarryFlag

the carry flag was mistakenly raised during a math operation

DIAG_E_MEM_NegativeFlag

the negative flag was mistakenly raised during a math operation

DIAG_E_MEM_OverflowFlag

the overflow flag was
mistakenly raised
during a math
operation

DIAG_E_MEM_SYS_STACK_OF

an overflow of the
system stack occurred

DIAG_E_MEM_SYS_STACK_UF

an underflow of the
system stack occurred

DIAG_E_MEM_SR0_TRAP

a SR0 trap occurred
and all interrupts have
been disabled

DIAG_E_MEM_CLASS_B_TRAP

a class B trap occurred
and all interrupts have
been disabled

DIAG_E_PWM_CURRENT_ZERO

Current measurement
greater zero during
startup phase

DIAG_E_PWM_CURRENT_OFFSET

Offset of current
measurement circuitry
on PWM output not
within limits

DIAG_E_PWM_LIMITS_RANGE

Pulse width not within
range on PWM output
(outside min/max
pulse)

<i>DIAG_E_PWM_LIMITS_TOL</i>	Pulse width not within tolerance window on PWM output
<i>DIAG_E_PWM_PERIOD_MISMATCH</i>	Period mismatch on PWM output
<i>DIAG_E_PWM_CURRENT</i>	Current not within limits on current controlled input
<i>DIAG_E_PWM_CURRENT_DEAD_TIME</i>	Set current not reached after dead time elapsed
<i>DIAG_E_PWM_CURRENT_OFFSETS_DRIFT</i>	Current offset too low (due to drift or HW defect)
<i>DIAG_E_PWD_LIMITS_FREQ</i>	Frequency limit error on timer input or counter error on incremental/counter input
<i>DIAG_E_PWD_LIMITS_PULSE_WIDTH</i>	Pulse width limit error on timer input
<i>DIAG_E_CYCLE_TIME</i>	Cycle time too high
<i>DIAG_E_RPP</i>	Insufficient gate drive on reverse polarity protection.
<i>DIAG_E_EXT_WD</i>	

	External WD has activated the safe state
<i>DIAG_E_LS_PROT</i>	Over-current condition on safety switch
<i>DIAG_E_OVD_STARTUP</i>	Over voltage startup test has failed
<i>DIAG_E_OVD</i>	Over voltage detection has activated the safe state
<i>DIAG_E_SAFETY_SW_INT</i>	Safety switch check error (internal switch)
<i>DIAG_E_SAFETY_SW_EXT</i>	Safety switch check error (external switch)
<i>DIAG_E_SAFETY_SW_SHUT_OFF</i>	Safety switch check failed during start-up
<i>DIAG_E_INVALID_DIAG_STATE</i>	Invalid diagnostic state
<i>DIAG_E_INVALID_STARTUP_STATE</i>	Invalid diagnostic state in startup state
<i>DIAG_E_INVALID_MAIN_STATE</i>	Invalid diagnostic state in main state
<i>DIAG_E_WD_STARTUP</i>	Watchdog startup test has failed

DIAG_E_SR_LowNibble

The feedback value of the low nibble of the shift register is faulty

DIAG_E_SR_HighNibble

The feedback value of the high nibble of the shift register is faulty

DIAG_E_FREQ_STARTUP

Internal clock frequency drifts from its normal value

DIAG_E_TIMEOUT

A timeout between Diagnostic Module and IO-Driver has occurred (f.ex driver functions of safety critical IO not called)

DIAG_E_APPL_SAFE_STATE

Application requested to activate the safe state

DIAG_E_PLL_VCO_NOT_LOCKED

The PLL/VCO lost its lock to the oscillator frequency.

DIAG_E_SW_INTERNAL

Internal SW error detected. Maybe SW or HW related.

DIAG_E_INIT_ERROR

Error during initialization of

diagnostic state
machine

DIAG_E_INT_WATCHDOG

Internal Watchdog has
not been serviced in
time - WD reset
occurred

DIAG_E_MEM_SOFTBREAK_TRAP

a SOFTBREAK trap
occurred (software
triggered)



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

[Enumerations](#) | [Functions](#)

DIAG_Functions.h

File Reference

Auxiliary functions for the diagnostic state machine. [More...](#)

```
#include "IO_Driver.h" #include "IO_Constants.h"
#include "DIAG_Constants.h"
```

Enumerations

```
enum DIAG_STARTUP_TEST_CTRL {
    DIAG_STARTUP_TEST_INHIBIT,
    DIAG_STARTUP_TEST_ACTIVATE
}
control modes for blocking startup tests More...
```

Functions

IO_ErrorType **DIAG_Status** (ubyte1 *diag_state,
DIAG_ERRORCODE *diag_error)
status function for diagnostic state machine

IO_ErrorType **DIAG_EnterSafestate** (void)

allows an application driven safe state

IO_ErrorType **DIAG_StartupTestCtrl**
(**DIAG_STARTUP_TEST_CTRL** ctrl)
Inhibits the startup tests.

IO_ErrorType **DIAG_EnableDischargeCircuit** (void)
Enables the usage of the integrated discharge
circuit (on ECU HW V5.00)

Detailed Description

Auxiliary functions for the diagnostic state machine.

Provides the interface to the diagnostic state machine

Enumeration Type Documentation

enum **DIAG_STARTUP_TEST_CTRL**

control modes for blocking startup tests

Enumerator:

DIAG_STARTUP_TEST_INHIBIT

inhibit startup tests

DIAG_STARTUP_TEST_ACTIVATE

activate startup
tests

Function Documentation

IO_ErrorType DIAG_EnableDischargeCircuit (void)

Enables the usage of the integrated discharge circuit (on ECU HW V5.00)

Returns

IO_ErrorTpye

Return values

IO_E_OK

Everything fine

IO_E_CHANNEL_BUSY

Discharge circuit has been already enabled

IO_E_DRIVER_INITIALIZED **IO_Driver_Init()** has been already called

Remarks

- This function has only influence on ECUs that provide an integrated discharge circuit (ECU HW V5.00).
- This function needs to be called before **IO_Driver_Init()** in order to enable the discharge circuit.

IO_ErrorType DIAG_EnterSafestate (void)

allows an application driven safe state

When this function is called the diagnostic state machine enters the safe state.

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_NOT_CONFIGURED

Diagnostic
state machine
still in init state

IO_E_ECU_ALREADY_IN_SAFE_STATE

ECU is already
in safe state

IO_ErrorType

DIAG_StartupTestCtrl (**DIAG_STARTUP_TEST_CTRL ctrl**)

Inhibits the startup tests.

Parameters

ctrl Control command for inhibiting/activating startup tests

Returns

IO_ErrorType

Return values

IO_E_OK

Everything fine

IO_E_INVALID_PARAMETER

Invalid parameter has been
passed to this function

IO_E_UNKNOWN

Function has not been
called before startup tests
are being executed

Remarks

This function needs to be called before the startup test are
being executed in order to inhibit them.

After the startup test are inhibited, this function has to be called once more in order to activate the startup tests.

```
IO_ErrorType  
DIAG_Status ( ubyte1 * diag_state,  
                DIAG_ERRORCODE * diag_error  
              )
```

status function for diagnostic state machine

Returns the current status as well as the error codes of the diagnostic state machine and the watchdog CPU.

Parameters

diag_state current state of the diagnostic state machine

diag_error error codes of the diagnostic state machine

Returns

IO_ErrorType

Return values

IO_E_NULL_POINTER null pointer has been passed

IO_E_OK everything fine

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

[Data Structures](#) | [Typedefs](#) | [Functions](#)

IO_ADC.h File Reference

IO Driver functions for ADC. [More...](#)

```
#include "IO_Driver.h"
```

Data Structures

struct **_io_adc_safety_conf**
Safety configuration for the ADC inputs. [More...](#)

Macros

Types of configurable Analog inputs

Input configuration for configurable ADC inputs. These defines can be used for the `mode` parameter of the function `IO_ADC_ChannelInit`.

```
#define IO_ADC_RATIOMETRIC 0x01
```

```
#define IO_ADC_CURRENT 0x03
```

```
#define IO_ADC_RESISTIVE 0x00
```

```
#define IO_ADC_ABSOLUTE 0x02
```

Range configuration for ADC inputs

Configuration of the ADC input range. These defines can be used for the `range` parameter of the function `IO_ADC_ChannelInit`.

```
#define IO_ADC_RANGE_5V 0
```

```
#define IO_ADC_RANGE_10V 1
```

Typedefs

typedef struct **_io_adc_safety_conf** **IO_ADC_SAFETY_CONF**
Safety configuration for the ADC inputs.

Functions

IO_ErrorType **IO_ADC_ChannelInit** (**IO_PIN** adc_channel, **ubyte1** mode, **ubyte1** range, const **IO_ADC_SAFETY_CONF** *const safety_conf)
Setup one ADC channel.

IO_ErrorType **IO_ADC_ChannelDeInit** (**IO_PIN** adc_channel)
Deinitializes one ADC input.

IO_ErrorType **IO_ADC_Get** (**IO_PIN** adc_channel, **ubyte2** *const adc_value, **bool** *const fresh)
Returns the value of the given ADC channel.

float4 **IO_ADC_BoardTempFloat** (**ubyte2** raw_value)
Calculates the board temperature in tenth degree Celsius.

sbyte2 **IO_ADC_BoardTempSbyte** (**ubyte2** raw_value)
Calculates the board temperature in degree Celsius.

Detailed Description

IO Driver functions for ADC.

Contains all service functions for the ADC.

There are four groups of Analog Inputs available:

- ADC 4-Mode: Can be configured as absolute, resistive (0-65535 Ohm), current (4-20mA) or ratiometric input. An additional sensor supply measurement (for the ratiometric case) will be configured and serves to correct the ADC signal. Input range is 0-5V for ratiometric and configurable (0-5V or 0-10V) for absolute mode. Up to 2 channels (4 channels for HY-TTC 32 variants) can be configured ([IO_ADC_00..IO_ADC_01](#) for HY-TTC 30 variants and [IO_ADC_14..IO_ADC_15](#) for HY-TTC 32 variants).
- ADC 3-Mode: 6 ADC inputs (4 ADC inputs for HY-TTC 32 variants) for current (4-20mA sensors), absolute and ratiometric measurement ([IO_ADC_10..IO_ADC_15](#) for HY-TTC 30 variants, [IO_ADC_10..IO_ADC_13](#) for HY-TTC 32 variants). Input range is 0-5V for ratiometric and configurable (0-5V or 0-10V) for absolute mode. If ratiometric mode is used the measurement of the sensor supply will be initialized.
- ADC 1-Mode: These inputs ([IO_ADC_20..IO_ADC_41](#)) can only be configured in absolute mode. Input range: 0-32V
- Normal ADC: Various ADC inputs for retrieving onboard voltages (UBat, Sensor-Supply, Board-Temperature, ...)

ADC Code Examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

Examples for ADC initialization:

```
// ADC 4-Mode:
IO_ADC_ChannelInit( IO_ADC_00
                    , IO_ADC_RESISTIVE // configuration for resistive
                    measurement        // parameter is ignored for
                    , 0                // no safety configuration
                    , IO_ADC_RESISTIVE
                    , NULL );

// ADC 4-Mode:
IO_ADC_ChannelInit( IO_ADC_01
                    , IO_ADC_ABSOLUTE // configuration for absolute
                    measurement        // measurement range 0-5V
                    , IO_ADC_RANGE_5V // no safety configuration
                    , NULL );

// ADC 3-Mode:
IO_ADC_ChannelInit( IO_ADC_10
                    , IO_ADC_CURRENT  // configuration for current
                    measurement        // parameter is ignored for
                    , 0                // no safety configuration
                    , IO_ADC_CURRENT
                    , NULL );

// Normal ADC:
IO_ADC_ChannelInit( IO_ADC_SENSOR_SUPPLY
                    , IO_ADC_ABSOLUTE // Only absolute allowed for
                    normal ADCs       // parameter is ignored for ADC
                    , 0                // channels other than 3-Mode and 2-Mode
                    , IO_ADC_CURRENT   // supported
                    , NULL );
```

Example for ADC task function call:

This function call is identical for every type of ADC inputs.

```
ubyte2 adc_val_0;
bool adc_fresh_0;

IO_ADC_Get( IO_ADC_00
            , &adc_val_0
            , &adc_fresh_0 );
```

Macro Definition Documentation

#define IO_ADC_ABSOLUTE 0x02

Absolute voltage measurement
use this configuration to measure an absolute voltage signal
Task function returns voltage in [mV]

#define IO_ADC_CURRENT 0x03

Current loop configuration
use this configuration if the connected sensor delivers a current signal (4..20mA sensors)
Task function returns current in [uA]

#define IO_ADC_RANGE_10V 1

ADC range 0 .. 10V

#define IO_ADC_RANGE_5V 0

ADC range 0 .. 5V

#define IO_ADC_RATIOMETRIC 0x01

Ratiometric configuration
use this configuration if the connected sensor is supplied by the sensor supply
([IO_ADC_SENSOR_SUPPLY](#)) and delivers a voltage signal.
Task function returns voltage in [mV]

#define IO_ADC_RESISTIVE 0x00

Resistive configuration
use this configuration if the sensor value shall be determined by measuring its resistance
Task function returns resistance in [Ohm]

Typedef Documentation

typedef struct _io_adc_safety_conf IO_ADC_SAFETY_CONF

Safety configuration for the ADC inputs.

Stores all relevant safety configuration parameters for the ADC inputs as an absolute value. The values shall be stated as mV, uA or Ohm - depending on the configuration of the channel.

It is not allowed to set the lower limit higher than or equal to the upper limit.

The valid min/max limits for each measured unit and range are like follows:

- `IO_ADC_ABSOLUTE` for 5V range configuration: 1mV ... 5000mV
- `IO_ADC_ABSOLUTE` for 10V range configuration: 1mV ... 10500mV
- `IO_ADC_ABSOLUTE` 32V inputs: 1mV ... 32780mV (1mV ... 33333mV for TTC32 variants)
- `IO_ADC_RATIOMETRIC`: 1mV ... 5000mV
- `IO_ADC_CURRENT`: 1uA ... 25000uA
- `IO_ADC_RESISTIVE`: 1 Ohm ... 65534 Ohm

The safety configuration is applicable for the following channels:

- `IO_ADC_00` .. `IO_ADC_01`
- `IO_ADC_10` .. `IO_ADC_15`
- `IO_ADC_20` .. `IO_ADC_21`

Function Documentation

float4 `IO_ADC_BoardTempFloat (ubyte2 raw_value)`

Calculates the board temperature in tenth degree Celsius.

The function converts the raw ADC value (retrieved from `IO_ADC_Get()`) to a temperature in degree Celsius and returns it as a float value.

Parameters

raw_value raw adc board temperature returned from the `IO_ADC_Get()` function

Returns

the board temperature in degree celsius (-55 .. 140.55 degree C)

Remarks

Usage:

```
IO_ADC_Get( IO_ADC_BOARD_TEMP, &raw_value, &fresh );
temp = IO_ADC_BoardTempFloat( raw_value );
```

sbyte2 `IO_ADC_BoardTempSbyte (ubyte2 raw_value)`

Calculates the board temperature in degree Celsius.

The function converts the raw ADC value (retrieved from `IO_ADC_Get()`) to a temperature in tenth degree Celsius and returns it as a sbyte1 value.

Parameters

raw_value raw adc board temperature returned from the `IO_ADC_Get()` function

Returns

returns the board temperature in tenth degree celsius (-550 .. 1405 which corresponds to -55 .. 140.5 degree C)

Remarks

- Usage:

```
IO_ADC_Get( IO_ADC_BOARD_TEMP, &raw_value, &fresh );
temp = IO_ADC_BoardTempSbyte( raw_value );
```

IO_ErrorType `IO_ADC_ChannelDeInit (IO_PIN adc_channel)`

Deinitializes one ADC input.

deinitializes the given ADC channel, allows reconfiguration by `IO_ADC_ChannelInit()`

Parameters

adc_channel ADC channel, one of:

- `IO_ADC_00 .. IO_ADC_01`
- `IO_ADC_10 .. IO_ADC_15`
- `IO_ADC_20 .. IO_ADC_41`
- `IO_K15`
- `IO_ADC_BOARD_TEMP`
- `IO_ADC_SENSOR_SUPPLY`
- `IO_ADC_UBAT`
- `IO_ADC_UBAT_CPU`
- `IO_ADC_NODE_ID_0`
- `IO_ADC_NODE_ID_1`
- `IO_INT_PIN_SHIFT_LB_HI`
- `IO_INT_PIN_SHIFT_LB_LO`
- `IO_INT_PIN_SHIFT1_LB_HI` (only relevant for HY-TTC 32 variants)
- `IO_INT_PIN_SHIFT1_LB_LO` (only relevant for HY-TTC 32 variants)

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	The given channel is not a analog input

Remarks

- The following channels form groups. A group always has to be configured as a whole.
 - `IO_ADC_00 .. IO_ADC_01`
 - `IO_ADC_10 .. IO_ADC_11`
 - `IO_ADC_12 .. IO_ADC_13`
 - `IO_ADC_14 .. IO_ADC_15`
 - `IO_ADC_20 .. IO_ADC_21`
 - `IO_ADC_22 .. IO_ADC_24`
 - `IO_ADC_25 .. IO_ADC_27`
 - `IO_ADC_30 .. IO_ADC_31`
 - `IO_ADC_32 .. IO_ADC_33`

```
IO_ErrorType IO_ADC_ChannelInit ( IO_PIN
                                ubyte1
                                ubyte1
                                const IO_ADC_SAFETY_CONF *const
                                )
                                adc_channel,
                                mode,
                                range,
                                safety_conf
```

Setup one ADC channel.

Parameters

adc_channel ADC channel, one of:

- `IO_ADC_00 .. IO_ADC_01`
- `IO_ADC_10 .. IO_ADC_15`
- `IO_ADC_20 .. IO_ADC_41`
- `IO_K15`

	<ul style="list-style-type: none"> • <code>IO_ADC_BOARD_TEMP</code> • <code>IO_ADC_SENSOR_SUPPLY</code> • <code>IO_ADC_5V2</code> • <code>IO_ADC_UBAT</code> • <code>IO_ADC_UBAT_CPU</code> • <code>IO_ADC_NODE_ID_0</code> • <code>IO_ADC_NODE_ID_1</code> • <code>IO_INT_PIN_SHIFT_LB_HI</code> • <code>IO_INT_PIN_SHIFT_LB_LO</code> • <code>IO_INT_PIN_SHIFT1_LB_HI</code> • <code>IO_INT_PIN_SHIFT1_LB_LO</code>
mode	Type of input: <ul style="list-style-type: none"> • <code>IO_ADC_RATIOMETRIC</code>: voltage measurement proportional to sensor supply voltage • <code>IO_ADC_CURRENT</code>: 0-27600uA input • <code>IO_ADC_RESISTIVE</code>: 0-65535Ohm input • <code>IO_ADC_ABSOLUTE</code>: normal voltage input
range	Measurement-range when mode is set to <code>IO_ADC_ABSOLUTE</code> . This parameter is only evaluated for analog channels with configurable range, i.e.: <ul style="list-style-type: none"> • <code>IO_ADC_00 .. IO_ADC_01</code> • <code>IO_ADC_10 .. IO_ADC_15</code> All other channels have a non-configurable range of 0-3V (parameter is ignored). The parameter can be one of: <ul style="list-style-type: none"> • <code>IO_ADC_RANGE_5V</code> • <code>IO_ADC_RANGE_10V</code>
safety_conf	Safety configuration.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the channel id does not exist
<code>IO_E_INVALID_PARAMETER</code>	parameter is out of range
<code>IO_E_CHANNEL_BUSY</code>	the ADC input channel is currently used by another task
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	The common driver init function has not been called
<code>IO_E_GROUP_CONFLICT</code>	desired configuration not possible due to a group conflict
<code>IO_E_TASK_NO_FREE_SLOTS</code>	No more free slots to setup task function
<code>IO_E_SW_INTERNAL</code>	Internal software error
<code>IO_E_CH_CAPABILITY</code>	The ADC capability of this channel has not been activated
<code>IO_E_SAFETY_NOT_SUPPORTED</code>	the given channel does not support safety features
<code>IO_E_DRV_SAFETY_CONF_NOT_CONFIGURED</code>	the driver has not been initialized as safety device - i.e. the safety feature is not available for this channel

Remarks

- The supported features depend on the selected channel:
 - `IO_ADC_00 .. IO_ADC_01`:
 - mode: `IO_ADC_ABSOLUTE`, `IO_ADC_RATIOMETRIC`, `IO_ADC_CURRENT` Or `IO_ADC_RESISTIVE`
 - `IO_ADC_10 .. IO_ADC_15`:
 - mode: `IO_ADC_ABSOLUTE`, `IO_ADC_RATIOMETRIC` Or `IO_ADC_CURRENT`
 - `IO_ADC_20 .. IO_ADC_41`:
 - mode: `IO_ADC_ABSOLUTE`

- `IO_K15`:
 - mode: `IO_ADC_ABSOLUTE`
 - `IO_ADC_5V2`:
 - mode: `IO_ADC_ABSOLUTE`
 - `IO_ADC_BOARD_TEMP`:
 - mode: `IO_ADC_ABSOLUTE`
 - `IO_ADC_SENSOR_SUPPLY`:
 - mode: `IO_ADC_ABSOLUTE`
 - `IO_ADC_UBAT`:
 - mode: `IO_ADC_ABSOLUTE`
 - `IO_ADC_UBAT_CPU`:
 - mode: `IO_ADC_ABSOLUTE`
 - `IO_ADC_NODE_ID_0/IO_ADC_NODE_ID_1`:
 - mode: `IO_ADC_ABSOLUTE`
 - `IO_INT_PIN_SHIFT_LB_LO/IO_INT_PIN_SHIFT_LB_HI/IO_INT_PIN_SHIFT1_LB_LO/IO_INT_PIN_SHIFT1_LB_HI`:
 - mode: `IO_ADC_RATIOMETRIC` (ratiometric measurement to internal 5V2_VM voltage)
- The following channels form groups. They have to be configured in the same mode within a group.
 - `IO_ADC_00 .. IO_ADC_01`
 - `IO_ADC_10 .. IO_ADC_11`
 - `IO_ADC_12 .. IO_ADC_13`
 - `IO_ADC_14 .. IO_ADC_15`
 - `IO_ADC_20 .. IO_ADC_21`
 - `IO_ADC_22 .. IO_ADC_24`
 - `IO_ADC_25 .. IO_ADC_27`
 - `IO_ADC_30 .. IO_ADC_31`
 - `IO_ADC_32 .. IO_ADC_33`
 - Check the alternate functions of the pins used in each group. A pin can only be configured for one a time and it has to be the same function within the group. The alternate functions can be found at

Remarks

- The channels `IO_ADC_UBAT`, `IO_ADC_UBAT_CPU`, `IO_ADC_BOARD_TEMP` and `IO_K15` initialized in the function `IO_Driver_Init`. Therefore the return value is `IO_E_CHANNEL_BUSY` if `IO_Driver_Init` has been called before.
- If the driver is initialized with safety-parameters, i.e., `IO_Driver_Init` has been called with a sa structure `!= NULL`, the channels `IO_ADC_5V2`, `IO_ADC_SENSOR_SUPPLY`, `IO_INT_PIN_SHIFT_LB_LO`, `IO_INT_PIN_SHIFT_LB_HI`, `IO_INT_PIN_SHIFT1_LB_LO` and `IO_INT_PIN_SHIFT1_LB_HI` are additionally initialized with the call to `IO_Driver_Init` (the call to `IO_ADC_Channellnit` will return `IO_E_CHANNEL_BUSY` if `IO_Driver_Init` has been called before).
- The parameter `range` is only valid if the mode is set to `IO_ADC_ABSOLUTE`
- For HY-TTC 32 and HY-TTC 32S channels `IO_ADC_14` and `IO_ADC_15` support also `IO_ADC_RES1`

```
IO_ErrorType IO_ADC_Get ( IO_PIN      adc_channel,
                          ubyte2 *const adc_value,
                          bool *const  fresh
                        )
```

Returns the value of the given ADC channel.

Parameters

adc_channel ADC channel, one of:

- `IO_ADC_00 .. IO_ADC_01`
- `IO_ADC_10 .. IO_ADC_15`
- `IO_ADC_20 .. IO_ADC_41`
- `IO_ADC_BOARD_TEMP`
- `IO_ADC_SENSOR_SUPPLY`
- `IO_ADC_5V2`
- `IO_ADC_UBAT`
- `IO_ADC_UBAT_CPU`
- `IO_K15`
- `IO_ADC_NODE_ID_0`
- `IO_ADC_NODE_ID_1`
- `IO_INT_PIN_SHIFT_LB_HI`
- `IO_INT_PIN_SHIFT_LB_LO`
- `IO_INT_PIN_SHIFT1_LB_HI` (only relevant for HY-TTC 32 variants)
- `IO_INT_PIN_SHIFT1_LB_LO` (only relevant for HY-TTC 32 variants)

adc_value ADC value, the range depends on the input group and its configuration (`type` parameter of `IO_ADC_ChannelInit()`):

- `IO_ADC_00 .. IO_ADC_01`
 - `IO_ADC_ABSOLUTE`: 0..10500 (0V..10.500V)
 - `IO_ADC_RATIOMETRIC`: 0..5000 (0V..5.000V)
 - `IO_ADC_CURRENT`: 0..27600 (0mA..27.600mA)
 - `IO_ADC_RESISTIVE`: 0..65535 (0Ohm..65535Ohm)
- `IO_ADC_10 .. IO_ADC_15`
 - `IO_ADC_ABSOLUTE`: 0..10500 (0V..10.500V)
 - `IO_ADC_RATIOMETRIC`: 0..5000 (0V..5.000V)
 - `IO_ADC_CURRENT`: 0..27600 (0mA..27.600mA)
- `IO_ADC_20 .. IO_ADC_21`
 - `IO_ADC_ABSOLUTE`: 0..32780 (0V..32.780V) (0V..33.333V for TTC32 variants)
- `IO_ADC_22 .. IO_ADC_27`
 - `IO_ADC_ABSOLUTE`: 0..32780 (0V..32.780V)
- `IO_ADC_28 .. IO_ADC_41`
 - `IO_ADC_ABSOLUTE`: 0..32780 (0V..32.780V) (0V..33.333V for TTC32 variants)
- `IO_ADC_BOARD_TEMP`
 - `IO_ADC_ABSOLUTE`: 4500..24055 (-60.17..148.62 degree C)
- `IO_ADC_5V2`
 - `IO_ADC_ABSOLUTE`: 0..5313 (0V..5.313V)
- `IO_ADC_SENSOR_SUPPLY`
 - `IO_ADC_ABSOLUTE`: 0..5313 (0V..5.313V)
- `IO_ADC_UBAT`
 - `IO_ADC_ABSOLUTE`: 0..55000 (0V..55.000V)
- `IO_ADC_UBAT_CPU`
 - `IO_ADC_ABSOLUTE`: 0..55000 (0V..55.000V)
- `IO_ADC_NODE_ID_0/IO_ADC_NODE_ID_1`
 - `IO_ADC_ABSOLUTE`: 0..32780 (0V..32.780V)
- `IO_K15`
 - `IO_ADC_ABSOLUTE`: 0..32780 (0V..32.780V)

fresh Status of the ADC value

- `TRUE`: ADC value is fresh (channel has been converted)

- `FALSE`: ADC value is old (channel has not been converted)

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_ADC_CHANNEL_STARTUP</code>	Channel is in initialization phase
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CH_CAPABILITY</code>	The given channel is not a analog input
<code>IO_E_NULL_POINTER</code>	A NULL pointer has been passed
<code>IO_E_FET_PROTECTION</code>	FET is disabled, protection is active
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	Channel has not been setup
<code>IO_E_ADC_INVALID</code>	The ADC value is invalid/not available

Remarks

- The temperature value in degree C must recalculated in the following manner, since the value has an offset of 10000 and is multiplied with 100:

$$\text{degree_value} = ((\text{float}) \text{adc_value} - 10000) / 100$$
Please use the functions `IO_ADC_BoardTempFloat()` or `IO_ADC_BoardTempSbyte()` for this calculation
- For the input `IO_ADC_00` .. `IO_ADC_01` and `IO_ADC_10` .. `IO_ADC_15` in `IO_ADC_CURRENT` mode, the inputs will be switched off for 2s if the current exceed 27.600mA. In this time the functions returns `IO_E_FET_PROTECTION`.
- For HY-TTC 32 and HY-TTC 32S channels `IO_ADC_14` and `IO_ADC_15` support also `IO_ADC_RESISTIVE` mode

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

[Data Structures](#) | [Typedefs](#) | [Functions](#)

IO_BRBL.h File Reference

API for accessing data in the branding block of the ECU. [More...](#)

```
#include "ptypes_xel67.h" #include "IO_Constants.h"
```

Data Structures

struct [_io_brbl_can_id_](#)
CAN ID structure. [More...](#)

struct [_io_brbl_can_param](#)
Branding block CAN parameter structure. [More...](#)

struct [_io_brbl_dids](#)
Entry definition for DID table. [More...](#)

Macros

```
#define IO\_BRBL\_XTEA\_PRIV\_KEY\_LEN 4U
```

```
#define IO\_BRBL\_XTEA\_PRIV\_KEY\_IDX\_0 0U  
Definitions for private key table.
```

```
#define IO\_BRBL\_XTEA\_PRIV\_KEY\_IDX\_1 1U
```

```
#define IO\_BRBL\_XTEA\_PRIV\_KEY\_IDX\_2 2U
```

```
#define IO\_BRBL\_XTEA\_PRIV\_KEY\_IDX\_3 3U
```

```
#define IO_BRBL_XTEA_PRIV_KEY_IDX_4 4U
```

```
#define IO_BRBL_XTEA_PRIV_KEY_IDX_5 5U
```

```
#define IO_BRBL_XTEA_PRIV_KEY_IDX_6 6U
```

```
#define IO_BRBL_XTEA_PRIV_KEY_IDX_7 7U
```

```
#define IO_BRBL_XTEA_PRIV_KEY_IDX_8 8U
```

```
#define IO_BRBL_XTEA_PRIV_KEY_IDX_9 9U
```

```
#define IO_BRBL_XTEA_PRIV_KEY_IDX_10 10U
```

```
#define IO_BRBL_XTEA_PRIV_KEY_IDX_11 11U
```

```
#define IO_BRBL_XTEA_PRIV_KEY_TBL_LEN 12U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_0 0U  
Definitions for custom DID table.
```

```
#define IO_BRBL_CUSTOM_DID_IDX_1 1U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_2 2U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_3 3U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_4 4U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_5 5U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_6 6U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_7 7U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_8 8U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_9 9U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_10 10U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_11 11U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_12 12U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_13 13U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_14 14U
```

```
#define IO_BRBL_CUSTOM_DID_IDX_15 15U
```

```
#define IO_BRBL_CUSTOM_DID_TBL_LEN 16U
```

Typedefs

```
typedef struct _io_brbl_can_id IO_BRBL_CAN_ID
```

CAN ID structure.

```
typedef struct _io_brbl_can_param IO_BRBL_CAN_PARAM
```

Branding block CAN parameter structure.

```
typedef struct _io_brbl_dids IO_BRBL_CUSTOM_DID
```

Entry definition for DID table.

Functions

```
IO_ErrorType IO_BRBL_Validate (void)
```

Validates the branding block.

```
IO_ErrorType IO_BRBL_GetXteaKey (ubyte1 key_num, ubyte4 *const  
key_buf, ubyte1 key_buf_len)
```

Returns a secret key from the secret key table that is located in the branding block.

```
IO_ErrorType IO_BRBL_GetDid (ubyte1 did_tbl_idx,  
IO_BRBL_CUSTOM_DID *const did_entry)
```

Returns a DID from the DID table that is located in the branding block.

```
IO_ErrorType IO_BRBL_GetCanParam (IO_BRBL_CAN_PARAM *const  
brbl_can_param)
```

Returns the CAN parameters set in the Branding Block.

Detailed Description

API for accessing data in the branding block of the ECU.

Read DataIdentifiers stored in the Branding Block

This section describes how to read a DataIdentifier that has been stored in the ECUs branding block.

```
IO_ErrorType rc_validate;
IO_ErrorType rc_get_did;
IO_ErrorType rc_get_can_params;
IO_BRBL_CUSTOM_DID did_item;
IO_BRBL_CAN_PARAM can_params;

// First the consistency of the BRBL has to be checked.
// The validity of
// the BRBL has to be checked only once per power cycle.
// It is best to
// call this function during the init phase (directly
// after calling \c IO_Driver_Init)
// because validating the BRBL will take approximately
// 1.2ms.
rc_brbl_validate = IO_BRBL_Validate();
if (rc_validate == IO_E_OK)
{
    rc_get_did = IO_BRBL_GetDid( IO_BRBL_CUSTOM_DID_IDX_0,
    &did_item );
    if (rc_get_did == IO_E_OK)
    {
        // The DID data can now be accessed as follows:
        // did_item.Did holds the DID (e.g. 0xFD00)
        // did_item.DidLength holds the length of the data
        // that is attached to the DID
        // did_item.DidData is a pointer to the DID data
        // (array with length did_item.DidLength)
        // Copying the DID data from the branding block
        // can be done as follows:
        for (i = 0; i < did_item.DidLength; i++)
        {
            app_did_data_buffer[i] = did_item.DidData[i];
        }
    }

    rc_get_can_params = IO_BRBL_GetCanParam( &can_params
    );
    if (rc_get_can_params == IO_E_OK)
```

```
    {  
        // The can parameters have been successfully read.  
    }  
}  
else  
{  
    // Reading from the BRBL is not possible because it  
    does  
}
```

Macro Definition Documentation

#define IO_BRBL_CUSTOM_DID_IDX_0 0U

Definitions for custom DID table.

Custom DID at index 0

#define IO_BRBL_CUSTOM_DID_IDX_1 1U

Custom DID at index 1

#define IO_BRBL_CUSTOM_DID_IDX_10 10U

Custom DID at index 10

#define IO_BRBL_CUSTOM_DID_IDX_11 11U

Custom DID at index 11

#define IO_BRBL_CUSTOM_DID_IDX_12 12U

Custom DID at index 12

#define IO_BRBL_CUSTOM_DID_IDX_13 13U

Custom DID at index 13

#define IO_BRBL_CUSTOM_DID_IDX_14 14U

Custom DID at index 14

#define IO_BRBL_CUSTOM_DID_IDX_15 15U

Custom DID at index 15

#define IO_BRBL_CUSTOM_DID_IDX_2 2U

Custom DID at index 2

#define IO_BRBL_CUSTOM_DID_IDX_3 3U

Custom DID at index 3

#define IO_BRBL_CUSTOM_DID_IDX_4 4U

Custom DID at index 4

#define IO_BRBL_CUSTOM_DID_IDX_5 5U

Custom DID at index 5

#define IO_BRBL_CUSTOM_DID_IDX_6 6U

Custom DID at index 6

#define IO_BRBL_CUSTOM_DID_IDX_7 7U

Custom DID at index 7

#define IO_BRBL_CUSTOM_DID_IDX_8 8U

Custom DID at index 8

#define IO_BRBL_CUSTOM_DID_IDX_9 9U

Custom DID at index 9

#define IO_BRBL_CUSTOM_DID_TBL_LEN 16U

Number of entries available in the DID table.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_0 0U

Definitions for private key table.

Reserved by ECU manufacturer!

#define IO_BRBL_XTEA_PRIV_KEY_IDX_1 1U

Application reprogramming key.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_10 10U

Key 10.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_11 11U

Key 11.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_2 2U

Key 2.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_3 3U

Key 3.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_4 4U

Key 4.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_5 5U

Key 5.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_6 6U

Key 6.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_7 7U

Key 7.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_8 8U

Key 8.

#define IO_BRBL_XTEA_PRIV_KEY_IDX_9 9U

Key 9.

#define IO_BRBL_XTEA_PRIV_KEY_LEN 4U

Length of private key in 32bit words

#define IO_BRBL_XTEA_PRIV_KEY_TBL_LEN 12U

Number of entries available in private key table.

Function Documentation

IO_ErrorType

IO_BRBL_GetCanParam (**IO_BRBL_CAN_PARAM** *const **brbl_can_param**)

Returns the CAN parameters set in the Branding Block.

Parameters

[out] **brbl_can_param** Branding block CAN parameter structure

Returns

IO_ErrorType

Return values

IO_E_OK

Everything ok.

IO_E_NULL_POINTER

A NULL pointer has been passed.

IO_E_CHANNEL_NOT_CONFIGURED

Branding block has not been validated successfully yet.

IO_ErrorType

IO_BRBL_GetDid (**ubyte1** **did_tbl_idx**,
IO_BRBL_CUSTOM_DID *const **did_entry**)

Returns a DID from the DID table that is located in the branding block.

Parameters

[in] **did_tbl_idx** Index within DID table to read from. Can be one of:
IO_BRBL_CUSTOM_DID_IDX_0 ..
IO_BRBL_CUSTOM_DID_IDX_15

[out] **did_entry** Structure that holds the DID number, the length of its data and a pointer to the DID data.

Returns

IO_ErrorType

Return values

IO_E_OK	Everything ok.
IO_E_NULL_POINTER	A NULL pointer has been passed.
IO_E_INVALID_PARAMETER	An invalid parameter has been passed (out of range).
IO_E_CHANNEL_NOT_CONFIGURED	Branding block has not been validated successfully yet.

```
IO_ErrorType IO_BRBL_GetXteaKey ( ubyte1      key_num,
                                  ubyte4 *const key_buf,
                                  ubyte1      key_buf_len
                                )
```

Returns a secret key from the secret key table that is located in the branding block.

Parameters

[in] key_num	Index within key table to read from. Can be one of: IO_BRBL_XTEA_PRIV_KEY_IDX_0 .. IO_BRBL_XTEA_PRIV_KEY_IDX_11
[out] key_buf	Array with 4 elements to store the key to.
[in] key_buf_len	Length of array passed for parameter <code>key_buf</code>

Returns

IO_ErrorType

Return values

IO_E_OK	Everything ok.
IO_E_NULL_POINTER	A NULL pointer has been passed.
IO_E_INVALID_PARAMETER	An invalid parameter has been passed (out of range).
IO_E_CHANNEL_NOT_CONFIGURED	Branding block has not been validated successfully yet.

Remarks

The secret key is returned encrypted. It can only be used in combination with the cipher functions in the module `IO_Crypt`.

IO_ErrorType IO_BRBL_Validate (void)

Validates the branding block.

Returns

IO_ErrorType

Return values

IO_E_OK	Everything ok.
IO_E_INVALID_CRC	Branding block inconsistent.
IO_E_UNKNOWN	Branding block does not fit to the installed bootloader.

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

[Data Structures](#) | [Macros](#) | [Typedefs](#) | [Functions](#)

IO_CAN.h File Reference

IO Driver functions for CAN communication. [More...](#)

```
#include "IO_Driver.h"
```

Data Structures

struct [_io_can_data_frame](#)
CAN data frame. [More...](#)

Macros

```
#define IO\_CAN\_BAUDRATE\_10K 0
```

```
#define IO\_CAN\_BAUDRATE\_20K 1
```

```
#define IO\_CAN\_BAUDRATE\_25K 2
```

```
#define IO\_CAN\_BAUDRATE\_50K 3
```

```
#define IO\_CAN\_BAUDRATE\_100K 4
```

```
#define IO\_CAN\_BAUDRATE\_125K 5
```

```
#define IO\_CAN\_BAUDRATE\_250K 6
```

```
#define IO_CAN_BAUDRATE_500K 7
```

```
#define IO_CAN_BAUDRATE_800K 8
```

```
#define IO_CAN_BAUDRATE_1000K 9
```

Message buffer direction

CAN Channel 0

uses the ECU pins C2 (CAN-H) and B2 (CAN-L)

CAN channel 1 (only available for HY-TTC 32 variants)

uses the ECU pins K3 (CAN-H) and J3 (CAN-L)

Selects the transmission direction of a CAN message buffer

```
#define IO_CAN_MSG_READ 0
```

```
#define IO_CAN_MSG_WRITE 1
```

CAN frame format

Selects the format for a CAN frame

```
#define IO_CAN_STD_FRAME 0
```

```
#define IO_CAN_EXT_FRAME 1
```

Typedefs

```
typedef struct _io_can_data_frame IO_CAN_DATA_FRAME  
CAN data frame.
```

Functions

IO_ErrorType **IO_CAN_Init** (**IO_PIN** channel, **ubyte1** baudrate)
Initialization of the CAN communication driver.

IO_ErrorType **IO_CAN_InitTimings** (**IO_PIN** channel, **ubyte1** brp, **bool** div8, **ubyte1** tseg1, **ubyte1** tseg2, **ubyte1** sjw)
Initialization of the CAN communication driver for given bit timings.

IO_ErrorType **IO_CAN_DeInitHandle** (**ubyte1** handle)
Deinitializes a single message handle.

IO_ErrorType **IO_CAN_DeInit** (**IO_PIN** channel)
Deinitializes the given CAN channel.

IO_ErrorType **IO_CAN_ConfigMsg** (**ubyte1** *const handle, **IO_PIN** channel, **ubyte1** mode, **ubyte1** id_format, **ubyte4** id, **ubyte4** ac_mask)
Configures a message object.

IO_ErrorType **IO_CAN_ReadMsg** (**ubyte1** handle, **IO_CAN_DATA_FRAME** *const buffer)
Returns the data of a message object.

IO_ErrorType **IO_CAN_WriteMsg** (**ubyte1** handle, const **IO_CAN_DATA_FRAME** *const data)
Transmits a CAN Message.

IO_ErrorType **IO_CAN_ConfigFIFO** (**ubyte1** *const handle, **IO_PIN** channel, **ubyte1** size, **ubyte1** mode, **ubyte1** id_format, **ubyte4** id, **ubyte4** ac_mask)
Configures a FIFO buffer.

IO_ErrorType **IO_CAN_ReadFIFO** (**ubyte1** handle, **IO_CAN_DATA_FRAME** *const buffer, **ubyte1** buffer_size, **ubyte1** *const rx_frames)
Reads the data from a FIFO buffer.

IO_ErrorType **IO_CAN_WriteFIFO** (**ubyte1** handle, const **IO_CAN_DATA_FRAME** *const data, **ubyte1** tx_length)
Writes CAN frames to a FIFO buffer.

IO_ErrorType **IO_CAN_Status** (**IO_PIN** channel, **ubyte1** *const rx_error_counter, **ubyte1** *const tx_error_counter)
Returns the error counters of the CAN channel.

IO_ErrorType **IO_CAN_MsgStatus** (**ubyte1** handle)
Returns the status of a message buffer object.

IO_ErrorType **IO_CAN_FIFOStatus** (**ubyte1** handle)
Returns the status of a FIFO buffer.

Detailed Description

IO Driver functions for CAN communication.

The CAN driver uses the MultiCAN module of the XC2000 CPU.

The CAN driver supports 1 CAN channel (2 CAN channels for HY-TTC 32 variants), so-called CAN nodes.

All CAN nodes share a common set of message objects, where each message object may be individually allocated to one of the CAN nodes. Besides serving as a storage container for incoming and outgoing frames, message objects may be combined to build gateways between the CAN nodes or to setup a FIFO buffer.

The message objects are organized in double chained lists, where each CAN node has its own list of message objects. A CAN node stores frames only into message objects that are allocated to the list of the CAN node. It only transmits messages from objects of this list.

The functions 'remote acknowledge' and 'remote request' are not supported.

Usage of the acceptance masks:

The acceptance mask defines the relevant bits of the CAN ID. A binary 1 marks a relevant bit in the CAN ID on the same position.

Setting all bits of the acceptance mask (0x1FFFFFFF) only accepts the ID set with the ID parameter and rejects all other IDs. Setting the acceptance mask to 0 causes the message buffer to accept any IDs.

Using this mechanism a message buffer can be used to accept a range of CAN IDs.

Example:

```
ac_mask = 0x1FFFFFF00 = 0 b 1 1111 1111 1111 1111
                        1111 0000 0000
id       = 0x00000200 = 0 b 0 0000 0000 0000 0000
                        0010 0000 0000
```

in this example all messages with an ID between 0x200 and 0x2FF are accepted.

CAN code examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

Examples for CAN initialization:

```
ubyte1 handle_w, handle_r, handle_fifo_w,
      handle_fifo_r;

IO_CAN_Init( IO_CAN_CHANNEL_0
            , IO_CAN_BAUDRATE_500K ); // Configure
            with 500 kbit/s

// standard message buffers //

IO_CAN_ConfigMsg( &handle_w
                 , IO_CAN_CHANNEL_0 // channel 0
                 , IO_CAN_MSG_WRITE // transmit
                 message buffer
                 , IO_CAN_STD_FRAME // standard ID
                 , 0
                 , 0);

IO_CAN_ConfigMsg( &handle_r
                 , IO_CAN_CHANNEL_0 // channel 0
                 , IO_CAN_MSG_READ  // receive
                 message buffer
                 , IO_CAN_STD_FRAME // standard ID
                 , 1
                 , 0x1FFFFFFF);      // accept only
ID 1

// FIFO message buffer //

IO_CAN_ConfigFIFO( &handle_fifo_w
                  , IO_CAN_CHANNEL_0 // channel 0
                  , 20                // 20 items
                  , IO_CAN_MSG_WRITE // transmit fifo
                  buffer
```

```

        , IO_CAN_STD_FRAME // standard ID
        , 0
        , 0 );

IO_CAN_ConfigFIFO( &handle_fifo_r
        , IO_CAN_CHANNEL_0 // channel 0
        , 20                // 20 items
        , IO_CAN_MSG_READ   // receive fifo
        buffer
        , IO_CAN_STD_FRAME // standard ID
        , 0
        , 0 );              // accept every
ID

```

Examples for CAN task function calls:

```

IO_CAN_DATA_FRAME can_frame;
IO_ErrorType can_status;

// check if new message has been received
can_status = IO_CAN_MsgStatus(handle_r);

if ((can_status == IO_E_OK) || (can_status ==
    IO_E_CAN_OVERFLOW))
{
    // if message has been received, read the
    // message from the buffer
    IO_CAN_ReadMsg(handle_r, &can_frame);

    // received message is now stored in can_frame
    // and can be used by the application
}

// assemble CAN frame:
can_frame.id = 1;
can_frame.id_format = IO_CAN_STD_FRAME;
can_frame.length = 4;
can_frame.data[0] = 1;
can_frame.data[1] = 2;
can_frame.data[2] = 3;
can_frame.data[3] = 4;

```

```
// transmit message
IO_CAN_WriteMsg(handle_w, &can_frame);

// wait until the transmission has been finished:
while (IO_CAN_MsgStatus(handle_w) != IO_E_OK);
```

Macro Definition Documentation

#define IO_CAN_BAUDRATE_1000K 9

Configure CAN Channel with 1000KBit/s. Used settings:
BRP = 2, TSEG1 = 16, TSEG2 = 3, SJW = 2 and DIV8 = FALSE
(Sampling Point 85%)

#define IO_CAN_BAUDRATE_100K 4

Configure CAN Channel with 100KBit/s. Used settings:
BRP = 25, TSEG1 = 13, TSEG2 = 2, SJW = 2 and DIV8 = FALSE
(Sampling Point 87.5%)

#define IO_CAN_BAUDRATE_10K 0

Configure CAN Channel with 10KBit/s. Used settings:
BRP = 25, TSEG1 = 16, TSEG2 = 3, SJW = 2 and DIV8 = TRUE
(Sampling Point 85%)

#define IO_CAN_BAUDRATE_125K 5

Configure CAN Channel with 125KBit/s. Used settings:
BRP = 20, TSEG1 = 13, TSEG2 = 2, SJW = 2 and DIV8 = FALSE
(Sampling Point 87.5%)

#define IO_CAN_BAUDRATE_20K 1

Configure CAN Channel with 20KBit/s. Used settings:
BRP = 10, TSEG1 = 16, TSEG2 = 8, SJW = 2 and DIV8 = TRUE

(Sampling Point 68%)

#define IO_CAN_BAUDRATE_250K 6

Configure CAN Channel with 250KBit/s. Used settings:
BRP = 10, TSEG1 = 13, TSEG2 = 2, SJW = 2 and DIV8 = FALSE
(Sampling Point 87.5%)

#define IO_CAN_BAUDRATE_25K 2

Configure CAN Channel with 25KBit/s. Used settings:
BRP = 10, TSEG1 = 16, TSEG2 = 3, SJW = 2 and DIV8 = TRUE
(Sampling Point 85%)

#define IO_CAN_BAUDRATE_500K 7

Configure CAN Channel with 500KBit/s. Used settings:
BRP = 5, TSEG1 = 13, TSEG2 = 2, SJW = 2 and DIV8 = FALSE
(Sampling Point 87.5%)

#define IO_CAN_BAUDRATE_50K 3

Configure CAN Channel with 50KBit/s. Used settings:
BRP = 50, TSEG1 = 13, TSEG2 = 2, SJW = 2 and DIV8 = FALSE
(Sampling Point 85%)

#define IO_CAN_BAUDRATE_800K 8

Configure CAN Channel with 800KBit/s. Used settings:
BRP = 2, TSEG1 = 16, TSEG2 = 8, SJW = 2 and DIV8 = FALSE
(Sampling Point 68%)

#define IO_CAN_EXT_FRAME 1

the ID parameter holds an extended (29-bit) ID

#define IO_CAN_MSG_READ 0

used to setup a message buffer for receiving

#define IO_CAN_MSG_WRITE 1

used to setup a message buffer for transmitting

#define IO_CAN_STD_FRAME 0

the ID parameter holds a standard (11-bit) ID

Typedef Documentation

typedef struct _io_can_data_frame IO_CAN_DATA_FRAME

CAN data frame.

Stores a data frame for the CAN communication.

Function Documentation

```
IO_ErrorType IO_CAN_ConfigFIFO ( ubyte1 *const handle,  
                                IO_PIN      channel,  
                                ubyte1      size,  
                                ubyte1      mode,  
                                ubyte1      id_format,  
                                ubyte4      id,  
                                ubyte4      ac_mask  
                                )
```

Configures a FIFO buffer.

Configures a FIFO buffer for the given CAN channel.

Parameters

handle Returns the FIFO buffer handle

channel CAN channel, one of:

- `IO_CAN_CHANNEL_0`
- `IO_CAN_CHANNEL_1`

Note

Second CAN Interface (`IO_CAN_CHANNEL_1`) is only available for HY-TTC 32 and HY-TTC 32S

Parameters

size Size of FIFO buffer (number of frames).

mode Mode for CAN Message, one of:

- `IO_CAN_MSG_READ`
- `IO_CAN_MSG_WRITE`

id_format Format of message identifier, one of:

- `IO_CAN_STD_FRAME`
- `IO_CAN_EXT_FRAME`

id CAN message identifier
ac_mask CAN acceptance mask, refer to [Usage of the acceptance mask](#) for further details.

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_CAN_MAX_MO_REACHED	no more HW message objects are available
IO_E_CAN_MAX_HANDLES_REACHED	no more free handles are available
IO_E_NULL_POINTER	null pointer has been passed
IO_E_INVALID_CHANNEL_ID	invalid channel number has been passed
IO_E_INVALID_PARAMETER	a parameter is out of range
IO_E_CHANNEL_NOT_CONFIGURED	channel has not been initialized
IO_E_CH_CAPABILITY	the given channel does not support the desired functionality

```
IO_ErrorType IO_CAN_ConfigMsg ( ubyte1 *const handle,  
                                IO_PIN      channel,  
                                ubyte1      mode,  
                                ubyte1      id_format,  
                                ubyte4      id,  
                                ubyte4      ac_mask  
                                )
```

Configures a message object.

Configures a message object for the given CAN channel and returns a message object handle.

Parameters

handle Returns the message object handle

channel CAN channel, one of:

- `IO_CAN_CHANNEL_0`
- `IO_CAN_CHANNEL_1`

Note

Second CAN Interface (`IO_CAN_CHANNEL_1`) is only available for HY-TTC 32 and HY-TTC 32S

Parameters

mode Mode for CAN Message, one of:

- `IO_CAN_MSG_READ`,
- `IO_CAN_MSG_WRITE`

id_format Format of message identifier, one of:

- `IO_CAN_STD_FRAME`,
- `IO_CAN_EXT_FRAME`

id CAN message identifier

ac_mask CAN acceptance mask, refer to **Usage of the acceptance mask** for further details.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CAN_MAX_MO_REACHED</code>	no more HW message objects are available
<code>IO_E_CAN_MAX_HANDLES_REACHED</code>	no more handles are available
<code>IO_E_NULL_POINTER</code>	null Pointer has been passed
<code>IO_E_INVALID_CHANNEL_ID</code>	invalid Channel ID has been passed

IO_E_INVALID_PARAMETER

invalid Parameter has been passed

IO_E_CHANNEL_NOT_CONFIGURED

the given channel was not initialized

IO_E_CH_CAPABILITY

the given channel does not support the desired functionality

IO_ErrorType IO_CAN_DeInit (IO_PIN channel)

Deinitializes the given CAN channel.

Allows re-initialization by `IO_CAN_Init()`

Parameters

channel CAN channel, one of:

- `IO_CAN_CHANNEL_0`
- `IO_CAN_CHANNEL_1`

Note

Second CAN Interface (`IO_CAN_CHANNEL_1`) is only available for HY-TTC 32 and HY-TTC 32S

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_INVALID_CHANNEL_ID

invalid parameter has been passed

IO_E_CHANNEL_NOT_CONFIGURED channel has not been initialized

IO_E_CH_CAPABILITY

the given channel does not support the desired functionality

IO_ErrorType IO_CAN_DeInitHandle (ubyte1 handle)

Deinitializes a single message handle.

Allows re-initialization by `IO_CAN_ConfigMsg()` or `IO_CAN_ConfigFIFO()`

Parameters

handle CAN message handle (retrieved from `IO_CAN_ConfigMsg()` or `IO_CAN_ConfigFIFO()`)

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_INVALID_PARAMETER	invalid parameter has been passed
IO_E_CHANNEL_NOT_CONFIGURED	handle has not been initialized

IO_ErrorType IO_CAN_FIFOStatus (ubyte1 handle)

Returns the status of a FIFO buffer.

If a CAN node is in bus-off state, the bus-off recovery sequence detection is started automatically. By default the CAN node is not allowed to continue bus communication once the bus-off recovery sequence has finished. It is necessary to call `IO_CAN_MsgStatus()` or `IO_CAN_FIFOStatus()` respectively to enable further bus communication.

Parameters

handle CAN message object handle (retrieved from `IO_CAN_ConfigFIFO()`)

Returns

IO_ErrorType

Return values

IO_E_OK
IO_E_BUSY

everything fine
transmission is ongoing.
Only reported for write
handle.

IO_E_CAN_OVERFLOW

Overflow of FIFO buffer
occurred. Data has been
lost. Only reported for
read handle.

IO_E_CAN_OLD_DATA

no new data has been
received. Only reported
for read handle.

IO_E_CAN_WRONG_HANDLE

invalid handle has been
passed

IO_E_CHANNEL_NOT_CONFIGURED the given handle has not
been configured

Remarks

- Calling this function also starts the Bus Off recovery sequence if the respective CAN node is in Bus Off.

```
IO_ErrorType IO_CAN_Init ( IO_PIN channel,  
                             ubyte1 baudrate  
                             )
```

Initialization of the CAN communication driver.

The function

- Enables the module
- Sets the module clock to 40MHz
- Automatically sets up the bit timing for the given baudrate

Parameters

channel CAN channel, one of:

- **IO_CAN_CHANNEL_0**
- **IO_CAN_CHANNEL_1**

Note

Second CAN Interface (IO_CAN_CHANNEL_1) is only available for HY-TTC 32 and HY-TTC 32S

Parameters

baudrate Baud rate

- IO_CAN_BAUDRATE_10K
- IO_CAN_BAUDRATE_20K
- IO_CAN_BAUDRATE_25K
- IO_CAN_BAUDRATE_50K
- IO_CAN_BAUDRATE_100K
- IO_CAN_BAUDRATE_125K
- IO_CAN_BAUDRATE_250K
- IO_CAN_BAUDRATE_500K
- IO_CAN_BAUDRATE_800K
- IO_CAN_BAUDRATE_1000K

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_INVALID_PARAMETER	invalid parameter has been passed
IO_E_INVALID_CHANNEL_ID	invalid channel ID has been passed
IO_E_CHANNEL_BUSY	channel has been initialized before
IO_E_CH_CAPABILITY	the given channel does not support the desired functionality

Remarks

Module is initialized only once. To re-initialize the module, the function **IO_CAN_DeInit()** needs to be called.

IO_ErrorType IO_CAN_InitTimings (IO_PIN **channel**,

```
    ubyte1 brp,  
    bool   div8,  
    ubyte1 tseg1,  
    ubyte1 tseg2,  
    ubyte1 sjw  
)
```

Initialization of the CAN communication driver for given bit timings.

The function

- Enables the module
- Sets the module clock to 40MHz
- Configures the CAN channel with the given bit timings

Parameters

channel CAN channel, one of:

- `IO_CAN_CHANNEL_0`
- `IO_CAN_CHANNEL_1`

Note

Second CAN Interface (`IO_CAN_CHANNEL_1`) is only available for HY-TTC 32 and HY-TTC 32S

Parameters

brp Baudrate prescaler (1 ... 63)

div8 Configures an additional clock divider of 8

tseg1 Time segment before sample point (3 ... 16)

tseg2 Time segment after sample point (2 ... 8)

sjw Synchronization jump width (1 ... 4)

Returns

`IO_ErrorType`

Return values

`IO_E_OK` everything fine

`IO_E_INVALID_PARAMETER` invalid parameter has been

	passed
IO_E_INVALID_CHANNEL_ID	invalid channel ID has been passed
IO_E_CHANNEL_BUSY	channel has been initialized before
IO_E_CH_CAPABILITY	the given channel does not support the desired functionality

Remarks

- The timing parameters and baudrate are calculated as follows:
The time quanta "tq" results from the Baudrate prescaler and the additional clock divider:

$$tq = brp / 40,000,000\text{Hz} \text{ , if div8 = FALSE}$$

$$tq = 8 * brp / 40,000,000\text{Hz} \text{ , if div8 = TRUE}$$

The synchronization time (Tsync [s]), Phase Buffer Segment Time 1 (Tseg1 [s]) and Phase Buffer Segment Time (Tseg2 [s]) are calculated as follows based upon "tq":

$$T_{sync} = 1 * tq$$

$$T_{Seg1} = seg1 * tq$$

$$T_{Seg2} = seg2 * tq$$

The overall bit time [s] and baudrate [bit/s] are calculated with:

$$bit_time = T_{sync} + T_{Seg1} + T_{Seg2}$$

$$baudrate = 1 / bit_time$$

Example: brp = 5, seg1 = 13, seg2 = 2, sjw = 2 and div8 = FALSE

$$tq = 5 / 40,000,000\text{Hz} = 125\text{ns}$$

$$T_{sync} = 1 * 125\text{ns} = 125\text{ns}$$

$$T_{Seg1} = 13 * 125\text{ns} = 1625\text{ns}$$

$$T_{Seg2} = 2 * 125\text{ns} = 250\text{ns}$$

$$bit_time = 125\text{ns} + 1625\text{ns} + 250\text{ns} = 2000\text{ns}$$

$$baudrate = 1/2000\text{ns} = 500,000 \text{ bit/s}$$

The sampling point can be calculated with:

$$\text{sampling_point} = (1 + \text{seg1}) / (1 + \text{seg1} + \text{seg2})$$

- Module is initialized only once. To re-initialize the module, the function `IO_CAN_DeInit()` needs to be called.

IO_ErrorType IO_CAN_MsgStatus (ubyte1 handle)

Returns the status of a message buffer object.

If a CAN node is in bus-off state, the bus-off recovery sequence detection is started automatically. By default the CAN node is not allowed to continue bus communication once the bus-off recovery sequence has finished. It is necessary to call `IO_CAN_MsgStatus()` or `IO_CAN_FIFOStatus()` respectively to enable further bus communication.

Parameters

handle CAN message object handle (retrieved from `IO_CAN_ConfigMsg()`)

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_BUSY	transmission is ongoing
IO_E_CAN_OVERFLOW	message object overflow
IO_E_CAN_OLD_DATA	no new data received
IO_E_CAN_WRONG_HANDLE	invalid handle has been passed
IO_E_CHANNEL_NOT_CONFIGURED	the given handle has not been configured

Remarks

- Calling this function also starts the Bus Off recovery sequence if the respective CAN node is in Bus Off.

IO_ErrorType

```
IO_CAN_ReadFIFO ( ubyte1 handle,  
                  IO_CAN_DATA_FRAME *const buffer,  
                  ubyte1 buffer_size,  
                  ubyte1 *const rx_frames  
                )
```

Reads the data from a FIFO buffer.

Copies received CAN frames from a given FIFO buffer to a SW frame buffer.

Parameters

- handle** CAN FIFO buffer handle (retrieved from `IO_CAN_ConfigFIFO()`)
- buffer** Pointer to SW frame buffer structure. The received frames will be stored there.
- buffer_size** Size of `buffer`, maximum number of frames to be written to the buffer. If `buffer_size` is higher than the actual size of the buffer, data outside the buffer will be overwritten!
- rx_frames** Number of valid frames which have been copied to `buffer`

Returns

`IO_ErrorType`

Return values

- | | |
|--|---|
| <code>IO_E_OK</code> | everything fine |
| <code>IO_E_NULL_POINTER</code> | null pointer has been passed to function |
| <code>IO_E_CAN_WRONG_HANDLE</code> | invalid handle has been passed |
| <code>IO_E_CHANNEL_NOT_CONFIGURED</code> | the given handler has not been configured |

IO_E_CAN_OVERFLOW

overflow of FIFO buffer occurred. Data has been lost. In rare cases received data might be in a wrong order. Please note that the function doesn't check whole buffer for overflow. For whole buffer overflow check please call `IO_CAN_FIFOStatus` function

IO_E_CAN_OLD_DATA

no new data is available

IO_ErrorType

```
IO_CAN_ReadMsg (ubyte1 handle,  
                IO_CAN_DATA_FRAME *const buffer  
                )
```

Returns the data of a message object.

Reads a message from a given message object. Returns whether the message is new or not.

Parameters

handle CAN message object handle (retrieved from `IO_CAN_ConfigMsg()`)

buffer Pointer to data buffer structure. The received frame will be stored there.

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything fine

`IO_E_CAN_OVERFLOW`

overflow of message object

IO_E_CAN_OLD_DATA	no new data has been received since the last read
IO_E_CAN_WRONG_HANDLE	invalid handle has been passed
IO_E_NULL_POINTER	null pointer has been passed
IO_E_CHANNEL_NOT_CONFIGURED	the given handle was not configured
IO_E_CAN_INVALID_DATA	the received data is invalid - read again to get the valid data

```
IO_ErrorType IO_CAN_Status ( IO_PIN      channel,
                             ubyte1 *const rx_error_counter,
                             ubyte1 *const tx_error_counter
                             )
```

Returns the error counters of the CAN channel.

Returns the transmit and receive error counters of the selected CAN channel.

Parameters

channel CAN channel, one of:

- **IO_CAN_CHANNEL_0**
- **IO_CAN_CHANNEL_1**

Note

Second CAN Interface (IO_CAN_CHANNEL_1) is only available for HY-TTC 32 and HY-TTC 32S

Parameters

rx_error_counter Value of the receive error counter

tx_error_counter Value of the transmit error counter

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_CAN_ERROR_PASSIVE	controller is in error passive state
IO_E_CAN_BUS_OFF	controller is in bus off state
IO_E_INVALID_CHANNEL_ID	wrong channel number has been passed
IO_E_NULL_POINTER	null pointer has been passed
IO_E_CHANNEL_NOT_CONFIGURED	the given channel has not been initialized
IO_E_CH_CAPABILITY	the given channel does not support the desired functionality

IO_ErrorType

```
IO_CAN_WriteFIFO ( ubyte1 handle,  
                   const IO_CAN_DATA_FRAME *const data,  
                   ubyte1 tx_length  
                   )
```

Writes CAN frames to a FIFO buffer.

Copies CAN Frames from a SW frame buffer to a transmit FIFO buffer. The CAN transmission will be started.

Parameters

- handle** CAN message object handle (retrieved from `IO_CAN_ConfigFIFO()`)
- data** Pointer to data structure. The data in this structure will be transmitted.
- tx_length** Number of frames in `data` which shall be transmitted.

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CAN_FIFO_FULL

FIFO is full - no data has been copied into FIFO

IO_E_CAN_WRONG_HANDLE

invalid handle has been passed

IO_E_NULL_POINTER

null pointer has been passed

IO_E_CHANNEL_NOT_CONFIGURED the given channel has not been configured

IO_ErrorType

```
IO_CAN_WriteMsg (ubyte1 handle,  
                 const IO_CAN_DATA_FRAME *const data  
                 )
```

Transmits a CAN Message.

Transmits a CAN message, using the given channel and message object. Returns whether the transmission has been started successfully or not.

Parameters

handle CAN message object handle (retrieved from `IO_CAN_ConfigMsg()`)

data Pointer to data structure. The data in this structure will be transmitted.

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_BUSY

message object busy -
no data has been
transmitted

IO_E_CAN_WRONG_HANDLE

invalid handle has been
passed

IO_E_NULL_POINTER

null pointer has been
passed

IO_E_CHANNEL_NOT_CONFIGURED the given handle has not
been configured

Main Page		Related Pages	Data Structures	Files
File List	Globals			
inc				

IO_Constants.h File Reference

Typedefs

Global defines for IO Driver. [More...](#)

```
#include "ptypes_xe167.h"
```

Macros

Error Values

Errors codes that a function might return in IO_ErrorType.

```
#define IO_E_OK 0
```

```
#define IO_E_BUSY 2
```

```
#define IO_E_UNKNOWN 3
```

```
#define IO_E_DRV_SAFETY_CONF_NOT_CONFIG 20
```

```
#define IO_E_INVALID_SAFETY_CONFIG 21
```

#define IO_E_SAFETY_NOT_SUPPORTED 22

#define IO_E_ECU_ALREADY_IN_SAFE_STATE 23

#define IO_E_INVALID_CRC 24

#define IO_E_NO_SAFETY_SWITCH_CONFIGURED 25

#define IO_E_DRIVER_INITIALIZED 26

#define IO_E_INVALID_DIAG_STATE 27

#define IO_E_NULL_POINTER 30

#define IO_E_INVALID_PARAMETER 31

#define IO_E_CHANNEL_BUSY 32

#define IO_E_CHANNEL_NOT_CONFIGURED 33

#define IO_E_INVALID_CHANNEL_ID 34

#define IO_E_FET_PROTECTION 35

#define IO_E_PERIODIC_NOT_CONFIGURED 36

#define IO_E_CH_CAPABILITY 37

#define IO_E_DRIVER_NOT_INITIALIZED 38

#define IO_E_GROUP_CONFLICT 39

#define IO_E_CAN_OVERFLOW 40

#define IO_E_CAN_WRONG_HANDLE 41

#define IO_E_CAN_MAX_MO_REACHED 42

#define IO_E_CAN_MAX_HANDLES_REACHED 43

#define IO_E_CAN_FIFO_FULL 44

#define IO_E_CAN_OLD_DATA 45

#define IO_E_CAN_ERROR_PASSIVE 46

#define IO_E_CAN_BUS_OFF 47

#define IO_E_CAN_INVALID_DATA 48

#define IO_E_WD_TRIGGER_DISABLED 50

#define IO_E_WD_TRIGGER_TEMPORARY_DISABLED 51

#define IO_E_WD_INT_ONLY_NON_SAFETY 52

#define IO_E_EEPROM_RANGE 60

#define IO_E_EEPROM_BUFFER_FULL 61

#define IO_E_EEPROM_CRC_MISMATCH 62

#define IO_E_UART_BUFFER_FULL 70

#define IO_E_UART_BUFFER_EMPTY 71

#define IO_E_UART_OVERFLOW 72

#define IO_E_UART_PARITY 73

#define IO_E_SPI_MAX_DEV_REACHED 80

#define IO_E_SPI_BUFFER_FULL 81

#define IO_E_ADC_INVALID 92

#define IO_E_ADC_CHANNEL_STARTUP 93

#define IO_E_PWM_CAPTURE_ERROR 100

#define IO_E_PWM_NOT_FINISHED 101

#define IO_E_PWM_OPEN_LOAD 102

#define IO_E_PWM_SHORT_CIRCUIT 103

#define IO_E_PWM_SHORT_BATTERY 104

#define IO_E_PWM_OPEN_LOAD_OR_SHORT_BATTERY 105

#define IO_E_PWM_CHANNEL_STARTUP 107

#define IO_E_PWM_OUTPUT_STARTUP_ERROR 108

#define IO_E_PWM_OUTPUT_DISABLED 109

#define IO_E_PWM_OUTPUT_LOW 110

#define IO_E_PWM_OUTPUT_HIGH 111

#define IO_E_PWM_DIAG_TRANSIENT_OSC 112

#define IO_E_PWM_CURRENT_INACCURATE 113

#define IO_E_PWD_TIMER_OVERFLOW 120

#define IO_E_PWD_HIGH_LEVEL 121

#define IO_E_PWD_LOW_LEVEL 122

#define IO_E_PWD_CAPTURE_ERROR 123

#define IO_E_PWD_NOT_FINISHED 124

#define IO_E_DO_CHANNEL_STARTUP 130

#define IO_E_DO_OUTPUT_STARTUP_ERROR 131

#define IO_E_DO_OPEN_LOAD 132

#define IO_E_DO_SHORT_CIRCUIT 133

#define IO_E_DO_SHORT_BATTERY 134

#define IO_E_DO_OPEN_LOAD_OR_SHORT_BATTERY 135

#define IO_E_DO_DIAG_TRANSIENT_OSC 136

#define IO_E_DO_OUTPUT_DISABLED 137

#define IO_E_DO_CURRENT_INACCURATE 138

#define IO_E_PROT_USER_OVERLOAD 140

#define IO_E_PROT_TEMP_OVERLOAD 141

#define IO_E_PROT_ACTIVE 142

#define IO_E_PROT_FATAL 143

#define IO_E_PROT_REENABLE 144

#define IO_E_PROT_PERMANENT_OFF 145

#define IO_E_DI_OPEN_LOAD 160

#define IO_E_DI_OPEN_LOAD_OR_SHORT_CIRCUIT 161

#define IO_E_DI_SHORT_CIRCUIT 162

#define IO_E_DI_SHORT_BATTERY 163

#define IO_E_DI_INVALID_VOLTAGE 164

#define IO_E_DI_INVALID_LIMITS 165

#define IO_E_NODEID_PINS_INVALID 170

#define IO_E_NODEID_EEPROM_MISMATCH 171

#define IO_E_NODEID_EEPROM_INVALID 172

#define IO_E_NODEID_EEPROM_FALLBACK 173

#define IO_E_PVG_SHORT_CIRCUIT 180

#define IO_E_PVG_SHORT_BATTERY 181

#define IO_E_PVG_OUTPUT_DISABLED 182

#define IO_E_VOUT_SHORT_CIRCUIT 190

#define IO_E_VOUT_SHORT_BATTERY 191

#define IO_E_VOUT_OUTPUT_DISABLED 192

#define IO_E_VOUT_PRECISION 193

#define IO_E_PID_NO_FREE_HANDLES 200

#define IO_E_PID_USED 201

#define IO_E_SW_INTERNAL 220

#define IO_E_SW_OUTPROT_SM 221

#define IO_E_WRONG_HW_TYPE 230

#define IO_E_TASK_NO_FREE_SLOTS 240

#define IO_E_DISCHARGE_FAILED 241

#define IO_E_RESET_COUNTER_INVALID 250

#define IO_E_SBRAM_CONTENT_INVALID 251

Typedefs

typedef uint8_t IO_ErrorType

Detailed Description

Global defines for IO Driver.

This header file defines the Error Codes for the IO-Driver.

Macro Definition Documentation

#define IO_E_ADC_CHANNEL_STARTUP 93

The given ADC channel is in its initialization phase and the low-pass filter of the analog input is still during tune-in. The initialization phase takes about 20ms. The ADC value which is returned is not valid.

#define IO_E_ADC_INVALID 92

The reported ADC value is invalid or not available. This error is reported in two cases. First, if an ADC value will be read out immediately after initializing the ADC channels and the ADC conversion of the respective channel has not been started yet (after startup). Secondly, this error is reported during runtime if a conversion error has been detected by the IO-Driver which means that the ADC did not convert the channel in the last conversion cycle.

#define IO_E_BUSY 2

Module or function is busy. This error is reported if a function or module has not yet finished its task. For example the EEPROM write function will return this error code if a previous write command has not been finished yet. Or if a channel is still initializing (e.g. during startup, changing modes, ...) and therefore not ready so far.

#define IO_E_CAN_BUS_OFF 47

The CAN node is in bus off state.
The bus-off recovery sequence is triggered by calling the function **IO_CAN_MsgStatus()** or **IO_CAN_FIFOStatus()**.

#define IO_E_CAN_ERROR_PASSIVE 46

The CAN node is in error passive state.

#define IO_E_CAN_FIFO_FULL 44

The FIFO buffer is full.
When reading: The respective FIFO buffer is full, but no data has been lost.
When writing: The data has not been accepted by the driver. The application needs to transmit it again when there is enough space in the buffer.

#define IO_E_CAN_INVALID_DATA 48

Received data is invalid.
Read again to get the valid data.

#define IO_E_CAN_MAX_HANDLES_REACHED 43

No more message handles are available.
The maximum number of message handles has been reached.
A message handle is generated every time the function

IO_CAN_ConfigMsg() or **IO_CAN_ConfigFIFO()** is called without returning an error.

#define IO_E_CAN_MAX_MO_REACHED 42

No more message objects are available.

The maximum number of available message objects has been reached. A single message object is needed to setting up a single message object with the function **IO_CAN_ConfigMsg()**. When configuring a FIFO buffer with the function **IO_CAN_ConfigFIFO()** the number of needed message objects equals the size of the FIFO buffer. (single message objects are joined together to a FIFO buffer)

#define IO_E_CAN_OLD_DATA 45

No new data is available.

This error is returned if no CAN frame has been received since the last successful read.

#define IO_E_CAN_OVERFLOW 40

Message object or FIFO buffer overflow.

This error is reported if CAN messages have been lost due to a full buffer. To avoid this error FIFO buffers can be used. If FIFO buffers are already used, try to increase the buffer size.

#define IO_E_CAN_WRONG_HANDLE 41

A wrong or invalid handle has been used.

This error is reported if:

- a non-existent handle has been used.
- if a write handle has been passed to a read function or vice versa
- if a message object handle has been passed to a FIFO function or vice versa

#define IO_E_CH_CAPABILITY 37

The IO channel (IO pin) does not support the requested feature. Two conditions can lead to this error code:

- For example when trying to initialize or use an ADC pin as PWM output.
- When trying to initialize an IO for a pin function which is not available on the ECU variant. (For example when trying to initialize a PWD input but it is not physically mounted on the used ECU variant)

#define IO_E_CHANNEL_BUSY 32

The IO channel (IO pin) is busy.

This error is reported if an IO Pin has been initialized before. To change the configuration of the channel during runtime the according De-Init function needs to be called before the channel can be again initialized with a new configuration.

#define IO_E_CHANNEL_NOT_CONFIGURED 33

The IO channel (IO pin) has not been initialized.
This error is reported by an IO driver task function if the channel has not been initialized. To initialize the channel, the according Init function needs to be called.

#define IO_E_DI_INVALID_LIMITS 165

The voltage thresholds passed to **IO_DI_Init** are not valid.

#define IO_E_DI_INVALID_VOLTAGE 164

A voltage level outside of the specified thresholds has been detected on a digital input.

#define IO_E_DI_OPEN_LOAD 160

An open load condition has been detected on a digital input.

#define IO_E_DI_OPEN_LOAD_OR_SHORT_CIRCUIT 161

An open load or short circuit to ground has has been detected on a digital input.

#define IO_E_DI_SHORT_BATTERY 163

An open load or short circuit to battery has has been detected on a digital input.

#define IO_E_DI_SHORT_CIRCUIT 162

A short circuit to ground has been detected on a digital input.

#define IO_E_DISCHARGE_FAILED 241

Discharging of capacitor failed.
Discharge circuit(new feature for ECU HW V5.00) was not able to discharge capacitor in time. Probably HW set-up regarding external safety switch is wrong (see also HW user manual).

#define IO_E_DO_CHANNEL_STARTUP 130

The digital output is in it's startup phase.
For digital output channels with current measurement the current measurement is being calibrated during this phase.

#define IO_E_DO_CURRENT_INACCURATE 138

Current measurement inaccurate.
This error is reported if the values for zero current compensation which are stored in the PDB are out of range or do not match the values which are measured during startup.

#define IO_E_DO_DIAG_TRANSIENT_OSC 136

The diagnostic functions on a digital output with analog feedback have been temporarily disabled.

PWM output in digital output mode:

This error code is reported if the output value of a digital output with analog feedback has been changed. Due to the low pass filter in the feedback path, the diagnostic functions are disabled for 100 ms after a change of the output level to avoid wrongly reported errors during the settling time of the filter.

Remarks

This is no error but a information for the application software that currently no output diagnostic is available on this channel. If this code is returned the application software should wait until the error **IO_E_OK** is returned before changing the output value again. Otherwise no diagnostic information can be provided.

#define IO_E_DO_OPEN_LOAD 132

An open load condition has been detected on a digital output.

#define IO_E_DO_OPEN_LOAD_OR_SHORT_BATTERY 135

An open load condition or a short circuit to battery voltage has been detected on a digital output.

#define IO_E_DO_OUTPUT_DISABLED 137

Digital outputs are disabled.

This error is reported if the high-side powerstages are disabled via **IO_POWER_Set** or not yet enabled after ECU startup. No diagnosis is possible during this stage.

#define IO_E_DO_OUTPUT_STARTUP_ERROR 131

The digital output could not be started up.
This error code is related to digital output channels with current measurement and describes that the offset of the current measurement is out of range. To protect the current measurement hardware (electric shunt) this output will stay disabled for the remaining driving cycle.

#define IO_E_DO_SHORT_BATTERY 134

A short circuit to battery voltage condition has been detected on a digital output.

#define IO_E_DO_SHORT_CIRCUIT 133

A short circuit to ground condition has been detected on a digital output.

#define IO_E_DRIVER_INITIALIZED 26

The common driver init function `IO_Driver_Init()` has been already called.
This error code is reported by a function if it is called after the common driver init function `IO_Driver_Init()` has been called.

#define IO_E_DRIVER_NOT_INITIALIZED 38

The common driver init function `IO_Driver_Init()` has not been called.

This error code is reported by the IO-Driver init functions if the common driver init function `IO_Driver_Init()` has not been called.

#define IO_E_DRV_SAFETY_CONF_NOT_CONFIG 20

Global safety configuration is missing.

This error is reported if an IO is defined as safety critical although no safety configuration has been passed to the `IO_Driver_Init()` (parameter `safety_conf`) function. An IO pin is considered as safety critical if a valid safety configuration has been passed to the init function (see `safety_conf` parameter of the functions `IO_ADC_ChannelInit()`, `IO_PWM_Init()`, `IO_PWD_IncInit()` and `IO_PWD_ComplexInit()`).

#define IO_E_ECU_ALREADY_IN_SAFE_STATE 23

The ECU is already in safe state.

This error code is reported by the function `DIAG_EnterSafestate` if the ECU is already in the safe state at the time the application requests to switch to the safe state.

#define IO_E_EEPROM_BUFFER_FULL 61

EEPROM buffer overrun.

An internal SPI buffer has reported an overrun, data was lost.

#define IO_E_EEPROM_CRC_MISMATCH 62

Error on CRC calculation.

The checksum stored in the EEPROM and the calculated one do not match.

The read data contains errors.

#define IO_E_EEPROM_RANGE 60

Invalid address range.

This error is reported if read or write operations are requested for non-existent EEPROM addresses.

#define IO_E_FET_PROTECTION 35

An internal switch (FET) has been disabled to protect the hardware from damages.

If the current on an internal FET is too high, the FET will be switched off by software to protect it from destruction. After 1s timeout the driver tries to re-enable the FET.

When a FET has been switched off by the protection mechanism, this error code will be returned by the respective task function. The measured values are therefore invalid and should not be used for further calculations.

#define IO_E_GROUP_CONFLICT 39

The IO channel (IO pin) cannot be configured due to conflicts with other IO channels.

This error is reported by initialization functions if the configuration cannot be performed due to conflicts with other already configured IO pins.

#define IO_E_INVALID_CHANNEL_ID 34

The IO channel (IO pin) does not exist.
This error is reported if a non-existent channel ID has been passed to the function.

#define IO_E_INVALID_CRC 24

CRC checksum wrong.
This error code is reported if the CRC calculation of the production data block has failed.

#define IO_E_INVALID_DIAG_STATE 27

The instruction is not permitted in the current diagnostic state.
This error code is reported if the instruction is not permitted in the current state of the diagnostic state machine (e.g. en-/disabling an IO Pin during `DIAG_STATE_STARTUP`).

#define IO_E_INVALID_PARAMETER 31

An invalid parameter has been passed to the function.
This error is reported if at least one of the parameters which have been passed to the function is outside the allowed range.

#define IO_E_INVALID_SAFETY_CONFIG 21

The safety configuration for the channel to be configured is invalid.
This error is reported if a parameter in the safety configuration

structure used for configuring a IO channel is wrong (see `safety_conf` parameter of the functions `IO_ADC_ChannelInit()`, `IO_PWM_Init()`, `IO_PWD_IncInit()` and `IO_PWD_ComplexInit()`).

#define IO_E_NO_SAFETY_SWITCH_CONFIGURED 25

No safety switch is configured.

This error is reported when trying to access (e.g. en- or disabling) a safety switch but no internal or external safety switch is configured.

#define IO_E_NODEID_EEPROM_FALLBACK 173

The bootloader used the (valid) values stored in the EEPROM to calculate the modifier as the voltage level on the pins `IO_PIN_K3` (`IO_ADC_NODE_ID_0`) and `IO_PIN_J3` (`IO_ADC_NODE_ID_1`) did not represent a valid modifier.

#define IO_E_NODEID_EEPROM_INVALID 172

The Node ID stored in the EEPROM was invalid on startup and overwritten with the Node ID determined via pins `IO_PIN_K3` and `IO_PIN_J3`.

#define IO_E_NODEID_EEPROM_MISMATCH 171

The Node ID stored in the EEPROM and the voltage levels on pins `IO_PIN_K3` and `IO_PIN_J3` do not match.

#define IO_E_NODEID_PINS_INVALID 170

The voltage levels on pins **IO_PIN_K3** (**IO_ADC_NODE_ID_0**) and **IO_PIN_J3** (**IO_ADC_NODE_ID_1**) represent no valid Node ID.

#define IO_E_NULL_POINTER 30

A NULL pointer has been passed to the function.
This error is reported if a non-optional pointer parameter of the function has been set to NULL.

#define IO_E_OK 0

everything is fine, no error has occurred.

#define IO_E_PERIODIC_NOT_CONFIGURED 36

The periodic interrupt timer has not been initialized.
This error code is reported if trying to disable the periodic interrupt timer although it has not been setup.

#define IO_E_PID_NO_FREE_HANDLES 200

No unused PID controllers available.
This error is reported when a PID controller is configured, but all available PID controller (**IO_PID_MAX_HANDLES** in total) are already used

#define IO_E_PID_USED 201

PID controller is currently in use.

This error is reported if a PID controller is de-initialized while it is in use. At present, this error code is not returned by any IO-Driver function.

#define IO_E_PROT_ACTIVE 142

An output has detected a overload situation and has been switched off to protect the hardware from damage.

High Side Outputs with Current Measurement:

If the current on a power output is too high, the output will be switched off by software to protect the system from thermal overload. The current limit for these outputs is 3.0A. The driver will switch off the output if the current is

- between 3.0A and 4.0A for more than 1s
- above 4.0A
- above "6*duty_cycle"A (value duty_cycle from 0..1, 50% duty_cycle means that the maximum current can be 3A, 25% duty cycle means 1.5A)

After a timeout of 1 second the driver tries to re-enable the output. When a output has been switched off by the protection mechanism, this error code will be returned by the respective task function.

High Side Outputs with Overcurrent Protection:

If the current on a power output is higher than

- 3.75A for longer than 1 second,
- 5A for longer than 250ms
- 7.1A for longer than 128ms

the output will be switched off by software to protect the system from thermal overload. After a timeout of 1

second the driver tries to re-enable the output (for the 7.1A limit the timeout is 10s). When an output has been switched off by the protection mechanism, this error code will be returned by the respective task function.

Low-Side Digital Outputs:

If the current on a low-side power output is too high, the output will be switched off by software to protect the system from thermal overload. The current limit for these outputs is 3.5A. The driver will switch off the affected low-side output if the current is:

- between 3.5A and 5.5A for more than 1s (see also [**IO_E_PROT_TEMP_OVERLOAD**](#)).
- above 5.5A

After a timeout of 1 second the driver tries to re-enable the output. When an output has been switched off by the protection mechanism, this error code will be returned by the respective task function.

Common for all power outputs

If the driver detects over temperature or loss of gate-drive of the reverse polarity protection the power outputs (Highside and Lowside) are switched off. During the time the outputs are switched off, this error code gets returned.

PVG/Voltage Outputs:

If the absolute value of the difference (U_{diff}) between the configured output voltage and the voltage measured with the analog feedback channel and is greater than 9.5V for more than 100ms, the output enters the protection state. $U_{diff} = (U_s - U_{fb})$, where U_s is the set output voltage and U_{fb} is the measured feedback voltage.

If $U_{diff} > 9.5V$ (e.g. short circuit to ground) the output is reduced to 25% for 1s.

If $U_{diff} < -9.5V$ (e.g. short circuit to battery) in the output is increased to 75% for 1s.

#define IO_E_PROT_FATAL 143

An output has been switched off due to a fatal overload condition to protect the hardware from damage.

PWM outputs:

If the current on these outputs rises above 5.4A, the output is switched off automatically. After a timeout of 1 second the driver tries to re-enable that output. When an output has been switched off by the protection-circuit, this error code will be returned by the respective task function.

High Side Outputs with Overcurrent Protection:

If the current on these output rises above 7.1A, output is switched off automatically. After a timeout of 10 seconds the driver tries to re-enable that output. When an output has been switched off by the protection-circuit, this error code will be returned by the respective task function.

Low-Side Digital Outputs:

If the current on these output rises above 7A, the protection-circuit gets activated and the affected output is switched off automatically. After a timeout of one second the driver tries to re-enable that output. When an output has been switched off by the protection-circuit, this error code will be returned by the respective task function.

PVG/Voltage Outputs:

These channels do not report fatal protection errors.

#define IO_E_PROT_PERMANENT_OFF 145

An output has been switched off permanently.

Reasons could be:

- Diagnostic module entered safe state
- ECU is being switched off via KL15-Hold by the application software

#define IO_E_PROT_REENABLE 144

An output which has been switched off has been re-enabled.

High Side Outputs with Current Measurement:

If these outputs are switched off due to an over-current condition, the IO-Driver will try to re-enable the output after 1 second. During the re-enable phase the IO-Driver returns this error code.

If the current is higher than the limits specified for the output, the power output will be switched off again during the re-enable phase.

High-Side Digital Outputs:

If these outputs are switched off due to an over-current condition, the IO-Driver will try to re-enable the output after 1 second. During the re-enable phase the IO-Driver returns this error code.

If the current is higher than the limits specified for the output, the power output will be switched off again during the re-enable phase.

Low-Side Digital Outputs:

If these outputs are switched off due to an over-current condition, the IO-Driver will try to re-enable the output after 1 second. During the re-enable phase the IO-Driver returns this error code.

If the current is higher than the limits specified for the

output, the power output will be switched off again during the re-enable phase.

PVG/Voltage Outputs:

If these outputs are switched off due to a large difference between the measured and configured voltage, the IO-Driver will try to re-enable the output after 1s.

#define IO_E_PROT_TEMP_OVERLOAD 141

An output has detected a temporary overload situation.

High Side Outputs with Current Measurement:

These channels can take 3.0A continuous current and up to 4A peak current for 1 second. If the current is above 3.0A the IO-Driver signals to the application with this error code that after 1 second the output stages will be switched off unless the current is decreasing to or below 3.0A.

If the power output has been switched off, the error code **IO_E_PROT_ACTIVE** will be reported.

High Side Outputs with Overcurrent Protection:

This error code is not returned by channels with slow overcurrent protection, as these outputs do allow a high current for 1 second. If the current is above 3.75A after 1 second the output stages will be switched off.

If the power output has been switched off, the error code **IO_E_PROT_ACTIVE** will be reported.

Low-Side Digital Outputs:

These channels can take 3.5A continuous current and up to 5.5A peak current for 1 second. If the current is above 3.5A the IO-Driver signals to the application with this error code that after 1 second the affected low-side output will be switched off unless the current is decreasing to or below

3.5A.

If the low-side output has been switched off, the error code **IO_E_PROT_ACTIVE** will be reported.

PVG/Voltage Outputs:

There is no temporary overload situation for PVG/Voltage outputs.

#define IO_E_PROT_USER_OVERLOAD 140

An output has detected a situation that was specified as overload by the user.

PWM Outputs:

If the measured current rises above the `overload_limit` specified upon initialization with `IO_PWM_Init`, the driver task-function will return this error code.

Digital Outputs:

If the measured current rises above the `overload_limit` specified upon initialization with `IO_DO_Init`, the driver task-function will return this error code.

PVG/Voltage Outputs:

For PVG/Voltage outputs there is no user configurable overload situation available.

#define IO_E_PVG_OUTPUT_DISABLED 182

PVG/Voltage outputs are disabled.

This error is reported if the PVG/Voltage Outputs are disabled via **IO_POWER_Set** or not yet enabled after ECU startup. No diagnosis is possible during this stage.

#define IO_E_PVG_SHORT_BATTERY 181

If the measured voltage on the analog feedback of a PVG output is above 95% of U_{Bat}, this error is reported by the step function.

#define IO_E_PVG_SHORT_CIRCUIT 180

If the measured voltage on the analog feedback of a PVG output is below 5% of U_{Bat}, this error is reported by the step function.

#define IO_E_PWD_CAPTURE_ERROR 123

A capture error occurred on a timer channel

PWM output with timer feedback in digital timer input mode:

This error is reported if two edges of the measured signal are too close to each other, and the internal timer cannot measure the time difference anymore (for example spikes caused by the switching of inductive loads in electric motors).

Digital Timer input:

This error is reported if two edges of the measured signal are too close to each other, and the internal timer cannot measure the time difference anymore (for example spikes caused by the switching of inductive loads in electric motors).

#define IO_E_PWD_HIGH_LEVEL 121

A constant high level has been detected on a timer channel.

PWM output with timer feedback in digital timer input mode:

This error is reported if no edges are captured for 100ms and a high level is detected on the input pin.

#define IO_E_PWD_LOW_LEVEL 122

A constant low level has been detected on a timer channel.

PWM output with timer feedback in digital timer input mode:

This error is reported if no edges are captured for 100ms and a low level is detected on the input pin.

#define IO_E_PWD_NOT_FINISHED 124

The timer channel has not yet finished the measurement.

Digital timer input in digital timer input mode:

Depending on the configuration a certain number of signal edges are required for a timing measurement. This error is reported if not all edges have been captured. If the task function is called multiple times during a period of the signal which shall be measured, the function will return **IO_E_OK** only if valid data is available, otherwise it returns this error code.

This error is also reported if no edges have been captured at all.

#define IO_E_PWD_TIMER_OVERFLOW 120

A timer overflow occurred.

This error code appears when the 24bit timer overflows. The time depends on the timer resolution which can be configured with the respective initialization function.

Remarks

timer_res = 0.2us -> max. period of input signal to measure
= 3.34s

timer_res = 0.4us -> max. period of input signal to measure
= 6.68s

timer_res = 0.8us -> max. period of input signal to measure
= 13.36s

timer_res = 1.6us -> max. period of input signal to measure
= 26.73s

timer_res = 3.2us -> max. period of input signal to measure
= 53.47s

#define IO_E_PWM_CAPTURE_ERROR 100

A capture error occurred on a PWM loop-back channel.

This error code can be the result of a stuck measurement timer or if two edges of the measured signal are too close to each other, and the internal timer cannot measure the time difference anymore.

PWM output with timer feedback in PWM output mode:

This error is reported if two edges of the measured signal are too close to each other, and the internal timer cannot measure the time difference anymore (for example spikes caused by the switching of inductive loads in electric motors).

#define IO_E_PWM_CHANNEL_STARTUP 107

The PWM output is in its startup phase.
For PWM channels with current measurement the current measurement is being calibrated during this phase.

#define IO_E_PWM_CURRENT_INACCURATE 113

Current measurement inaccurate.
This error is reported if the values for zero current compensation which are stored in the PDB are out of range or do not match the values which are measured during startup.

#define IO_E_PWM_DIAG_TRANSIENT_OSC 112

The diagnostic functions on a PWM output with analog feedback have been temporarily disabled.
This error code is reported if a error is detected on a PWM output with disabled diagnostic margin. Due to the low pass filter in the feedback path, the diagnostic functions are disabled for 50ms after a change of the output level to avoid wrongly reported errors during the settling time of the filter.

Remarks

This is no error but a information for the application software that currently no output diagnostic is available on this channel. If this code is returned the application software should wait until the error **IO_E_OK** is returned before changing the output value again. Otherwise no diagnostic information can be provided.

#define IO_E_PWM_NOT_FINISHED 101

The timer feedback channel of a PWM output has not yet finished the pulse-width and period measurement. This error code is currently being suppressed and only used internally. Currently it will not be returned by any API function

#define IO_E_PWM_OPEN_LOAD 102

An open load condition has been detected. This error is reported if the output signal cannot be measured via the timer feedback and the level on the analog feedback channel corresponds to the device-internal pull-up resistor.

Remarks

No open load detection is available if the power stages have been disabled.

#define IO_E_PWM_OPEN_LOAD_OR_SHORT_BATTERY 105

An open load or short to battery condition has been detected. At the point of time when this error is returned, it was not yet possible to distinguish between open load or short to battery. After 50ms – if the error is still present – depending on the voltage level at the output pin the following errors will be returned:

- **IO_E_PWM_SHORT_BATTERY** if voltage is near battery voltage
- **IO_E_PWM_OPEN_LOAD** if voltage is in open load range
- Still **IO_E_PWM_OPEN_LOAD_OR_SHORT_BATTERY** if the IO driver was not able to precisely determine which error occurred.

#define IO_E_PWM_OUTPUT_DISABLED 109

PWM outputs are disabled.

This error is reported if the high-side powerstages are disabled via **IO_POWER_Set** or not yet enabled after ECU startup. No diagnosis is possible during this stage.

#define IO_E_PWM_OUTPUT_HIGH 111

A high level has been detected on a PWM channel.

This error is returned, if the diagnostic margin of a PWM channel has been deactivated and the duty cycle exceeds the higher diagnostic margin of 250us. In this state – due to the high duty cycle – a short to battery situation cannot be distinguished from a faultless situation.

Remarks

This is no error, but an information for the application software that due to the high duty cycle only reduced diagnosis is available.

#define IO_E_PWM_OUTPUT_LOW 110

A low level has been detected on a PWM channel.

This error is returned, if the diagnostic margin of a PWM channel has been deactivated and the duty cycle exceeds the lower diagnostic margin of 100us. In this state – due to the low duty cycle – a short to ground situation cannot be distinguished from a faultless situation.

Remarks

This is no error, but an information for the application software that due to the low duty cycle only reduced diagnosis is available.

#define IO_E_PWM_OUTPUT_STARTUP_ERROR 108

The PWM output has been switched off because a error occurred during it's startup phase.
For PWM channels with current measurement this means that the offset of the current measurement is out of range. For reasons to protect the current measurement hardware (electric shunt) this output will stay disabled for the remaining driving cycle.

#define IO_E_PWM_SHORT_BATTERY 104

A short circuit to battery condition has been detected.
This error is reported if the output signal cannot be measured via the timer feedback and the level on the analog feedback channel is larger than 0.8 times UBAT.

Remarks

No short circuit detection is available if the power stages have been disabled. Refer to the **ECU power functions** for further details.

#define IO_E_PWM_SHORT_CIRCUIT 103

A short circuit condition has been detected.
This error is reported if the output signal cannot be measured via the timer feedback and the level on the feedback channel is low (ground level) This means that no edges could be captured

on the timer feedback channel within a timeout of 4 times of the PWM period time.

The error condition is reset as soon as the timer was able to capture a complete PWM period (3 edges).

Remarks

No short circuit detection is available if the power stages have been disabled. Refer to the [ECU power functions](#) for further details.

#define IO_E_RESET_COUNTER_INVALID 250

Reset counter is invalid The content of the reset save CPU registers that are used to store the reset counter is inconsistent.

#define IO_E_SAFETY_NOT_SUPPORTED 22

The given channel does not support to be configured safety critical.

This error is reported if an IO channel is configured as safety critical but does not support this feature (reported by [IO_ADC_ChannelInit\(\)](#), [IO_PWM_Init\(\)](#)).

#define IO_E_SBRAM_CONTENT_INVALID 251

Content of standby memory invalid The content of a section of the standby memory is inconsistent and can't be read.

#define IO_E_SPI_BUFFER_FULL 81

SPI hardware buffer is full.

This error code is used internally by the SPI driver. It is not reported by any API function.

#define IO_E_SPI_MAX_DEV_REACHED 80

Maximum number of SPI devices reached.

This error code is used internally by the SPI driver. It is not reported by any API function.

#define IO_E_SW_INTERNAL 220

Internal error has occurred.

This error can be caused by a SW or processor malfunction.

#define IO_E_SW_OUTPROT_SM 221

An internal error within the output protection has occurred. An error within the state machine which handles the output protection for power outputs with current measurement occurred. This is an error caused by SW malfunction.

#define IO_E_TASK_NO_FREE_SLOTS 240

Maximum number of tasks reached.

Certain IOs use a task running in the background for managing protection mechanisms or executing PID controllers. If the number of free task slots reaches its maximum, this error code

gets returned. The root cause of such a error is most probably a malfunction within the IO driver software.

#define IO_E_UART_BUFFER_EMPTY 71

A UART software buffer is empty.

This error code is used internally by the UART driver. It is not reported by any API function.

#define IO_E_UART_BUFFER_FULL 70

The UART software buffer is full.

When receiving: Too much data has been received since the last successful read operation - data has been lost.

When transmitting: The given data does not fit into the buffer, data has been rejected. Try again when there is more space in the buffer.

#define IO_E_UART_OVERFLOW 72

Overflow in the UART hardware buffer.

The hardware buffer reported an overflow. This happens if too much data has been received between two consecutive SW cycles.

To avoid this problem the application can call the function **IO_UART_Task()** at any time. This function copies the data from the hardware buffer to the software buffer.

#define IO_E_UART_PARITY 73

UART parity error.

The received parity bit doesn't match the calculated one.

#define IO_E_UNKNOWN 3

General error. No further information can be provided.

#define IO_E_VOUT_OUTPUT_DISABLED 192

PVG/Voltage outputs are disabled.

This error is reported if the PVG/Voltage Outputs are disabled via **IO_POWER_Set** or not yet enabled after ECU startup. No diagnosis can be applied while the output stage is disabled.

#define IO_E_VOUT_PRECISION 193

Voltage on the output is outside the allowed tolerance.

If the configured output voltage is not reached after 150ms within a tolerance of (+/- 200mV), this error is returned by the step-function.

#define IO_E_VOUT_SHORT_BATTERY 191

A short circuit to battery condition has been detected on a voltage output.

This error is reported if after the settling time of 150ms the configured voltage is not reached and the the voltage measured on the analog feedback is greater than (UBat - 1000mV).

#define IO_E_VOUT_SHORT_CIRCUIT 190

A short circuit to ground condition has been detected on a voltage output.

This error is reported if after the settling time of 150ms the configured voltage is not reached and the voltage measured on the analog feedback is less than 1000mV.

#define IO_E_WD_INT_ONLY_NON_SAFETY 52

Configuring/triggering/disabling of internal watchdog via API functions is only possible if IO Driver is configured as non-safety relevant. If IO Driver is configured as safety relevant configuration and triggering/disabling is performed via functions **IO_Driver_Init()** / **IO_Driver_TaskBegin()** automatically

#define IO_E_WD_TRIGGER_DISABLED 50

The trigger mechanism for the external watchdog has been disabled.

This error code is returned when the diagnostic part of the IO-Driver has activated the Safe-State!

#define IO_E_WD_TRIGGER_TEMPORARY_DISABLED 51

The trigger mechanism for the external watchdog has been temporary disabled for diagnosis of the watchdog.

#define IO_E_WRONG_HW_TYPE 230

Hardware type does not match.

This error is caused by using an IO-Driver for the wrong device (e.g. TTC-30XH driver for TTC-30XI).

Attention

The IO-Driver remains uninitialized and no out- or inputs will work.

Typedef Documentation

typedef **ubyte2** **IO_ErrorType**

Every driver function returns an error code of type **IO_ErrorType**. Refer to **Error Values** for a documentation of the possible values

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

IO_Crypt.h File Reference

Functions

API for I/O driver cryptographic functions. [More...](#)

```
#include "ptypes_xe167.h" #include "IO_Constants.h"
```

Macros

```
#define IO_CRYPT_XTEA_KEY_LEN 4U
```

Definition of key length for XTEA algorithm.

Functions

IO_ErrorType **IO_Crypt_GetPseudoRandomNumber** (**ubyte4** *const prn)
Returns a pseudo random number.

IO_ErrorType **IO_Crypt_XteaEncipher** (**ubyte4** *const v0, **ubyte4** *const v1, const **ubyte4** *const key, **bool** key_encrypted)
Enciphers a 64bit value with the XTEA algorithm (in-place)

IO_ErrorType IO_Crypt_XteaDecipher (**ubyte4** *const v0, **ubyte4** *const v1, const **ubyte4** *const key, **bool** key_encrypted)
Deciphers a 64bit value with the XTEA algorithm (in-place)

IO_ErrorType IO_Crypt_XteaEncipher32 (**ubyte4** *const v, const **ubyte4** *const key, **bool** key_encrypted)
Enciphers a 32bit value with the XTEA algorithm (in-place)

IO_ErrorType IO_Crypt_XteaDecipher32 (**ubyte4** *const v, const **ubyte4** *const key, **bool** key_encrypted)
Deciphers a 32bit value with the XTEA algorithm (in-place)

Detailed Description

API for I/O driver cryptographic functions.

XTEA block cipher algorithm

The following section shows the XTEA block cipher algorithm as C code examples. The examples can be used as basis for integration of the algorithm into a diagnostic tester. The tester only needs the functions for enciphering (a comparable decipher function is used by the ECU). Nevertheless, for the purpose of completion both encipher and decipher functions are listed. The XTEA algorithm is a freely available algorithm for symmetrical encryption purposes. The current cipher implementation uses 64 feistel rounds. The algorithm uses a 128bit secret key that can be chosen freely but must of course be kept secret. If the chosen secret key becomes public security is compromised!

The application code does not have to implement the following functions because the algorithm is already included in the I/O driver library:

- `IO_Crypt_XteaEncipher`
- `IO_Crypt_XteaDecipher`
- `IO_Crypt_XteaEncipher32`
- `IO_Crypt_XteaDecipher32`

XTEA block cipher algorithm details

```
#define XTEA_DELTA          (ubyte4)0x9E3779B9UL
#define XTEA_DELTA_SHORT    (ubyte2)0x79B9U
#define XTEA_NUM_CYCLES     (ubyte4)32UL // 1
                                cycle corresponds to 2 feistel rounds. 32
                                cycles := 64 feistel rounds
```

```
// Enciphers a 32bit value
void xtea_encipher_u32( ubyte4 * const v
                        , const ubyte4 * const key
                        )
{
    ubyte2 i; // use a machine word for the loop
              variable
    ubyte2 sum;
    ubyte2 v0;
    ubyte2 v1;

    sum = 0;

    v0 = (ubyte2)*v;
    v1 = (ubyte2)(*v >> 16);

    for (i = 0; i < XTEA_NUM_CYCLES; i++) // 1
        cycle corresponds to 2 feistel rounds. 32
        cycles := 64 feistel rounds
    {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^
            (sum + (ubyte2)key[sum & 3]);
        sum += XTEA_DELTA_SHORT; // adding XTEA
        delta value. overrun of sum is irrelevant
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^
            (sum + (ubyte2)key[(sum>>11) & 3]);
    }

    *v = (ubyte4)v0;
    *v |= (ubyte4)v1 << 16;
}
```

```

}

// Deciphers a 32bit value
void xtea_decipher_u32( ubyte4 * const v
                        , const ubyte4 * const key
                        )
{
    ubyte2 i; // use a machine word for the loop
               variable
    ubyte2 sum;
    ubyte2 v0;
    ubyte2 v1;

    sum = (ubyte2)((ubyte4)XTEA_DELTA_SHORT *
                  (ubyte4)XTEA_NUM_CYCLES); // overrun of sum
                                             is irrelevant

    v0 = (ubyte2)*v;
    v1 = (ubyte2)(*v >> 16);

    for (i = 0; i < XTEA_NUM_CYCLES; i++) // 1
                                             cycle corresponds to 2 feistel rounds. 32
                                             cycles := 64 feistel rounds
    {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^
              (sum + (ubyte2)key[(sum>>11) & 3]);
        sum -= XTEA_DELTA_SHORT;
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^
              (sum + (ubyte2)key[sum & 3]);
    }

    *v = (ubyte4)v0;
    *v |= (ubyte4)v1 << 16;
}

```

```

// Enciphers a 64bit value (2*32bit, high word in
    v1, low word in v0)
void xtea_encipher( ubyte4 * const v0
                   , ubyte4 * const v1

```

```

, const ubyte4 * const key )
{
    ubyte2 i; // use a machine word for the loop
    variable
    ubyte4 sum;

    sum = 0;

    for (i = 0; i < XTEA_NUM_CYCLES; i++) // 1
        cycle corresponds to 2 feistel rounds. 32
        cycles := 64 feistel rounds
    {
        *v0 += (((*v1 << 4) ^ (*v1 >> 5)) + *v1)
        ^ (sum + key[sum & 3]);
        sum += XTEA_DELTA; // overrun of sum is
        irrelevant
        *v1 += (((*v0 << 4) ^ (*v0 >> 5)) + *v0)
        ^ (sum + key[(sum>>11) & 3]);
    }
}

```

```

// Deciphers a 64bit value (2*32bit, high word in
    v1, low word in v0)
void xtea_decipher( ubyte4 * const v0
                    , ubyte4 * const v1
                    , const ubyte4 * const key )
{
    ubyte2 i; // use a machine word for the loop
    variable
    ubyte4 sum;

    sum = (ubyte4)((ubyte8)XTEA_DELTA *
        (ubyte8)XTEA_NUM_CYCLES); // overrun of sum
    is irrelevant

    for (i = 0; i < XTEA_NUM_CYCLES; i++) // 1
        cycle corresponds to 2 feistel rounds. 32
        cycles := 64 feistel rounds
    {

```

```
        *v1 -= (((*v0 << 4) ^ (*v0 >> 5)) + *v0)
^ (sum + key[(sum>>11) & 3]);
        sum -= XTEA_DELTA;
        *v0 -= (((*v1 << 4) ^ (*v1 >> 5)) + *v1)
^ (sum + key[sum & 3]);
    }
}
```

Macro Definition Documentation

#define IO_CRYPT_XTEA_KEY_LEN 4U

Definition of key length for XTEA algorithm.

The XTEA algorithm operates with 128 bit keys (4*32bit)

Function Documentation

IO_ErrorType

IO_Crypt_GetPseudoRandomNumber (**ubyte4** *const **prn**)

Returns a pseudo random number.

Returns

IO_ErrorType

Return values

IO_E_OK

Everything ok.

IO_E_NULL_POINTER

A NULL pointer has been passed.

IO_E_CHANNEL_NOT_CONFIGURED I/O driver has not been initialized before.

Remarks

This functions returns a 32bit random number. If a 64bit random number is needed, the function can be called twice like in code example below.

```
ubyte8 prn64;  
ubyte4 prn32;  
  
IO_Crypt_GetPseudoRandomNumber (&prn32) ;  
prn64 = (ubyte8)prn32 << 32;  
IO_Crypt_GetPseudoRandomNumber (&prn32) ;  
prn64 |= prn32;
```

IO_ErrorType

IO_Crypt_XteaDecipher (**ubyte4** *const **v0**,

```

        ubyte4 *const v1,
        const ubyte4 *const key,
        bool key_encrypted
    )

```

Deciphers a 64bit value with the XTEA algorithm (in-place)

Parameters

[in, out]	v0	lower half of 64bit block to decipher
[in, out]	v1	upper half of 64bit block to decipher
[in]	key	secret key, must have 4 elements
[in]	key_encrypted	set to TRUE if given key is encrypted (e.g. because it originates from the branding block), otherwise to FALSE .

Returns

IO_ErrorType

Return values

IO_E_OK Everything ok.
IO_E_NULL_POINTER A NULL pointer has been passed.

IO_ErrorType

```

IO_Crypt_XteaDecipher32 ( ubyte4 *const v,
                          const ubyte4 *const key,
                          bool key_encrypted
                          )

```

Deciphers a 32bit value with the XTEA algorithm (in-place)

Parameters

[in, out]	v	32bit block to decipher
[in]	key	secret key, must have 4 elements
[in]	key_encrypted	set to TRUE if given key is encrypted (e.g. because it originates from the branding block), otherwise to FALSE .

Returns

IO_ErrorType

Return values

IO_E_OK	Everything ok.
IO_E_NULL_POINTER	A NULL pointer has been passed.

Remarks

The XTEA algorithm has been designed to work on 64bit blocks (2x32). This 32bit version is a derivative of it that operates based on the same secret key but using only 4x16bit of it instead of 4x32bit.

IO_ErrorType

```
IO_Crypt_XteaEncipher ( ubyte4 *const      v0,  
                        ubyte4 *const      v1,  
                        const ubyte4 *const key,  
                        bool                key_encrypted  
                        )
```

Enciphers a 64bit value with the XTEA algorithm (in-place)

Parameters

[in, out]	v0	lower half of 64bit block to encipher
[in, out]	v1	upper half of 64bit block to

		encipher
[in]	key	secret key, must have 4 elements
[in]	key_encrypted	set to TRUE if given key is encrypted (e.g. because it originates from the branding block), otherwise to FALSE .

Returns

IO_ErrorType

Return values

IO_E_OK	Everything ok.
IO_E_NULL_POINTER	A NULL pointer has been passed.

IO_ErrorType

```
IO_Crypt_XteaEncipher32 ( ubyte4 *const      v,
                          const ubyte4 *const key,
                          bool                key_encrypted
                          )
```

Enciphers a 32bit value with the XTEA algorithm (in-place)

Parameters

[in, out]	v	32bit block to encipher
[in]	key	secret key, must have 4 elements
[in]	key_encrypted	set to TRUE if given key is encrypted (e.g. because it originates from the branding block), otherwise to FALSE .

Returns

IO_ErrorType

Return values

IO_E_OK

Everything ok.

IO_E_NULL_POINTER A NULL pointer has been passed.

Remarks

The XTEA algorithm has been designed to work on 64bit blocks (2x32). This 32bit version is a derivative of it that operates based on the same secret key but using only 4x16bit of it instead of 4x32bit.

[Main Page](#)[Related Pages](#)[Data Structures](#)[Files](#)[File List](#)[Globals](#)

inc >

[Data Structures](#) | [Typedefs](#) | [Functions](#)

IO_DIO.h File Reference

IO Driver functions for Digital Input/Output. [More...](#)

```
#include "IO_Driver.h"
```

Data Structures

struct [_io_driver_di_limits](#)

Voltage limits for digital inputs. [More...](#)

Macros

Pull up / down configuration

Pull up/down resistor for the digital inputs

```
#define IO\_DI\_PU 0x01
```

```
#define IO\_DI\_PD 0x02
```

Typedefs

typedef struct [_io_driver_di_limits](#) [IO_DRIVER_DI_LIMITS](#)

Voltage limits for digital inputs.

Functions

IO_ErrorType **IO_DI_Init** (**IO_PIN** di_channel, **ubyte1** pupd, const **IO_DRIVER_DI_LIMITS** *const limits)
Setup the Digital Inputs.

IO_ErrorType **IO_DO_Init** (**IO_PIN** do_channel, **ubyte2** overload_limit)
Setup the Digital Outputs.

IO_ErrorType **IO_DI_Delinit** (**IO_PIN** di_channel)
Deinitializes a DI channel.

IO_ErrorType **IO_DO_Delinit** (**IO_PIN** do_channel)
Deinitializes a DO channel.

IO_ErrorType **IO_DI_Get** (**IO_PIN** di_channel, **bool** *const di_value)
Returns the state of a Digital Input.

IO_ErrorType **IO_DO_Set** (**IO_PIN** do_channel, **bool** do_value, **ubyte2** *const voltage_fb)
Sets the state of a Digital Output.

IO_ErrorType **IO_DO_GetCur** (**IO_PIN** do_channel, **ubyte2** *const current, **bool** *const fresh)
Returns the measured current of the given channel.

Detailed Description

IO Driver functions for Digital Input/Output.

Contains all service functions for the digital in/outputs.

DIO Code Examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

DIO initialization examples:

```
IO_DRIVER_DI_LIMITS limits = { 0, 3000, 3000, 32000
    };

IO_DI_Init( IO_DI_00, IO_DI_PU, &limits );
    // configure digital input with pull-up

IO_DO_Init( IO_DO_00, 2500 );
    // digital output with 2500mA overload
    configuration

IO_POWER_Set( IO_INT_POWERSTAGE_ENABLE, IO_POWER_ON
    );    // enable high-side outputs
```

DIO task function examples:

```
bool di_val_0;
ubyte2 do_voltage_fb;

IO_DI_Get( IO_DI_00                // read value of
    digital input
    , &di_val_0 );

IO_DO_Set( IO_DO_00                // set digital
    output value and get analog feedback
    , TRUE
    , &do_voltage_fb );
```

Macro Definition Documentation

#define IO_DI_PD 0x02

pull-down resistor

#define IO_DI_PU 0x01

pull-up resistor

Typedef Documentation

typedef struct `_io_driver_di_limits` `IO_DRIVER_DI_LIMITS`

Voltage limits for digital inputs.

Contains the thresholds for valid low- and high-levels for digital inputs. The range for the low-level is defined by the voltages `low_thresh1` and `low_thresh2`, where `low_thresh1` is the lower limit for a low-level and `low_thresh2` the upper limit.

The range for the high-level is defined by the voltages `high_thresh1` and `high_thresh2`, where `high_thresh1` is the lower limit for a high-level and `high_thresh2` the upper limit.

The value of `low_thresh1` must always be smaller than `low_thresh2` and `high_thresh1` must always be smaller than `high_thresh2`.

It is possible to configure a hysteresis by setting `low_thresh2` bigger than `high_thresh1`. In this mode an already detected logic level will be hold as long as the analog voltage varies within the hysteresis band `low_thresh2 - high_thresh1`. Please see the following examples:

Examples:

```
// voltage limits without hysteresis
IO_DRIVER_DI_LIMITS limits1 = { 0, 2000, 3000, 5000
    };
```

In the above example `limits1` defines the range 0-2000mV as valid low-level and 3000-5000mV as valid high-level. The voltage range between 2000 and 3000mV represents an invalid area. In this case the error code `IO_E_DI_INVALID_VOLTAGE` will be returned.

```
// voltage limits with hysteresis
```

```
IO_DRIVER_DI_LIMITS limits2 = { 0, 3000, 2000, 5000  
    };
```

In the above example `limits1` defines the range 0-2000mV as valid low-level and 3000-5000mV as valid high-level. The range 2000-3000mV forms a hysteresis. If the voltage value rises or drops within this band (between 2000 and 3000mV) the former state of the channel will be held.

Function Documentation

IO_ErrorType IO_DI_DeInit (IO_PIN di_channel)

Deinitializes a DI channel.

Parameters

di_channel Digital input:

- `IO_DI_00..IO_DI_07`
- `IO_DI_10..IO_DI_31`

Returns

IO_ErrorType:

Return values

IO_E_OK

everything fine

IO_E_INVALID_CHANNEL_ID

the given core id does not exist

IO_E_CHANNEL_NOT_CONFIGURED the given channel is not configured

IO_E_CH_CAPABILITY

The DI capability of this channel has not been activated

Remarks

- The following channels form groups. A group always has to be configured as a whole.
 - `IO_DI_00 .. IO_DI_01`
 - `IO_DI_02 .. IO_DI_03`
 - `IO_DI_04 .. IO_DI_05`
 - `IO_DI_06 .. IO_DI_07`
 - `IO_DI_10 .. IO_DI_11`
 - `IO_DI_12 .. IO_DI_13`
 - `IO_DI_14 .. IO_DI_15`
 - `IO_DI_16 .. IO_DI_18`

◦ IO_DI_19 .. IO_DI_21

```
IO_ErrorType IO_DI_Get ( IO_PIN      di_channel,  
                        bool *const di_value  
                        )
```

Returns the state of a Digital Input.

Parameters

di_channel Digital input:

- IO_DI_00 .. IO_DI_07
- IO_DI_10 .. IO_DI_31

di_value Input value:

- TRUE: High Level
- FALSE: Low Level

Returns

IO_ErrorType

Return values

IO_E_OK

IO_E_INVALID_CHANNEL_ID

IO_E_CH_CAPABILITY

IO_E_NULL_POINTER

IO_E_CHANNEL_NOT_CONFIGURED

IO_E_ADC_INVALID

IO_E_DI_SHORT_CIRCUIT

everything fine

the channel id
does not exist

The given
channel is not a
digital input

null pointer
passed as
argument

the given
channel is not
configured

ADC
measurement
error

short to ground

IO_E_DI_SHORT_BATTERY

was detected
short to battery
was detected

IO_E_DI_OPEN_LOAD

open-load
situation was
detected

IO_E_DI_INVALID_VOLTAGE

voltage
measured not
within configured
voltage limits

IO_E_DI_OPEN_LOAD_OR_SHORT_CIRCUIT

open-load or
short to GND
detected (not
distinguishable)

Remarks

The digital input is determined according to the following rules and order:

1. If the measured voltage lies between `low_thresh1` and `low_thresh2` configured with `IO_DI_Init`, `FALSE` is reported as digital value. If the voltage lies between `high_thresh1` and `high_thresh2`, `TRUE` is reported as digital value. In both cases, **IO_E_OK** is returned as error-code.
2. If a pull-up is configured on the input and the voltage lies between 4.75V and 5.5V, the value in `di_value` is not valid and the function returns **IO_E_DI_OPEN_LOAD**.
3. If a pull-down or no pull-resistor is configured on the input and the voltage lies between 0V and 1.25V, the value in `di_value` is not valid and the function returns **IO_E_DI_OPEN_LOAD_OR_SHORT_CIRCUIT**.
4. If the measured voltage lies between `UBat` and `UBat - 1.25V`, `TRUE` is stored in `di_value` and the function returns **IO_E_DI_SHORT_BATTERY**.
5. In all other cases, the value in `di_value` is not valid and the function returns **IO_E_DI_INVALID_VOLTAGE**, as the measured voltage neither lies in the range defined by the user, nor in any range that allows diagnosis.

Note

The voltage levels defined by the user via the parameter `limits` of `IO_DI_Init` always have priority over diagnosis functionality. For example, if an input is configured with pull-up resistor and the voltage range 0-6V is defined as valid low-level, no open-load error will be returned by this function.

IO_ErrorType

```
IO_DI_Init ( IO_PIN          di_channel,
             ubyte1          pupd,
             const IO_DRIVER_DI_LIMITS *const limits
           )
```

Setup the Digital Inputs.

Parameters

di_channel Digital input:

- `IO_DI_00..IO_DI_07`
- `IO_DI_10..IO_DI_31`

pupd Pull-up/down configuration, one of (only for `IO_DI_02..IO_DI_07`):

- `IO_DI_PU`: configure with pull-up resistor
- `IO_DI_PD`: configure with pull-down resistor

limits voltage limits for low/high-levels. If `NULL`, default limits will be used. See `IO_DRIVER_DI_LIMITS` for details.

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the channel id does not exist
<code>IO_E_CHANNEL_BUSY</code>	the digital output channel is currently used by another function
<code>IO_E_INVALID_PARAMETER</code>	parameter is out of range

IO_E_GROUP_CONFLICT	desired configuration not possible due to a group conflict.
IO_E_CH_CAPABILITY	The DI capability of this channel has not been activated
IO_E_DI_INVALID_LIMITS	the given voltage limits are not valid
IO_E_DRIVER_NOT_INITIALIZED	the common driver init function <code>IO_Driver_Init()</code> has not been called

Remarks

- If the parameter `limits` is `NULL`, default values will be used:
 - low-level: 0..1500mV
 - high-level: 3500..32000mV
- The parameter `pupd` is only considered for digital inputs which allow the configuration of a pull-up/down resistor; i.e. **IO_DI_02..IO_DI_07**. For all other digital inputs this parameter is ignored.
- All digital inputs are connected via multiplexers to the CPU. A changing value on the connector pin may only be available after some delay.
- The following channels form groups. They have to be configured in the same mode within a group.
 - **IO_DI_00 .. IO_DI_01**
 - **IO_DI_02 .. IO_DI_03**
 - **IO_DI_04 .. IO_DI_05**
 - **IO_DI_06 .. IO_DI_07**
 - **IO_DI_10 .. IO_DI_11**
 - **IO_DI_12 .. IO_DI_13**
 - **IO_DI_14 .. IO_DI_15**
 - **IO_DI_16 .. IO_DI_18**
 - **IO_DI_19 .. IO_DI_21**
- Check the alternate functions of the pins used in each group. A pin can only be configured for one function at a time and it has to be

the same function within the group. The alternate functions can be found at [IO_Pins.h](#)

IO_ErrorType IO_DO_DeInit (IO_PIN do_channel)

Deinitializes a DO channel.

Parameters

do_channel Digital Output:

- [IO_DO_00..IO_DO_07](#)
- [IO_DO_10..IO_DO_11](#)
- [IO_DO_20..IO_DO_25](#)
- [IO_DO_30..IO_DO_35](#)

Returns

[IO_ErrorType](#):

Return values

[IO_E_OK](#)

everything fine

[IO_E_INVALID_CHANNEL_ID](#)

the given core id does not exist

[IO_E_CHANNEL_NOT_CONFIGURED](#) the given channel is not configured

[IO_E_CH_CAPABILITY](#)

The DO capability of this channel has not been activated

Remarks

- The following channels form groups. A group always has to be configured as a whole.
 - [IO_DO_00](#), [IO_DO_20](#)
 - [IO_DO_01](#), [IO_DO_21](#)
 - [IO_DO_02](#), [IO_DO_22](#)
 - [IO_DO_03](#), [IO_DO_23](#)
 - [IO_DO_04](#), [IO_DO_24](#)
 - [IO_DO_05](#), [IO_DO_25](#)

```
IO_ErrorType IO_DO_GetCur ( IO_PIN      do_channel,
                           ubyte2 *const current,
                           bool *const  fresh
                           )
```

Returns the measured current of the given channel.

Parameters

- do_channel** Digital Output:
- `IO_DO_20..IO_DO_25`
- current** Measured current
- Range: 0..7575 (0mA..7575mA)
- fresh** Fresh flag for current value
- `TRUE` value is valid
 - `FALSE` no new value available

Returns

`IO_ErrorType`:

Return values

- | | |
|--|--|
| <code>IO_E_OK</code> | everything fine |
| <code>IO_E_CHANNEL_NOT_CONFIGURED</code> | the given channel is not configured |
| <code>IO_E_INVALID_CHANNEL_ID</code> | the given channel id does not exist |
| <code>IO_E_ADC_INVALID</code> | an ADC error occurred |
| <code>IO_E_PWM_CURRENT_INACCURATE</code> | Current measurement has reduced accuracy |
| <code>IO_E_NULL_POINTER</code> | A NULL pointer has been passed to the function |
| <code>IO_E_CH_CAPABILITY</code> | The DO capability of this channel has not been activated |

```
IO_ErrorType IO_DO_Init ( IO_PIN do_channel,
```

ubyte2 overload_limit

)

Setup the Digital Outputs.

Parameters

do_channel

Digital Output:

- **IO_DO_00..IO_DO_07 Digital High-Side Outputs with analog feedback**
- **IO_DO_10..IO_DO_11 Digital Low-Side with analog feedback**
- **IO_DO_20..IO_DO_25 Digital High-Side Outputs with analog and current feedback**
- **IO_DO_30..IO_DO_35 Push-Pull Digital Outputs**

overload_limit

Configurable limit [mA] above which

IO_E_PROT_USER_OVERLOAD is reported.

Parameter is ignored for **IO_DO_10..IO_DO_11** and

IO_DO_30 .. IO_DO_35

- overload_limit configured to 0: overload limit disabled; **IO_E_PROT_USER_OVERLOAD** error will not be reported
- overload_limit configured to 1mA .. 2999mA: **IO_E_PROT_USER_OVERLOAD** error will be reported via function **IO_DO_Set()** if the measured current is above defined overload_limit
- if measured current exceeds 3000mA other error codes than **IO_E_PROT_USER_OVERLOAD** are reported (see error codes of function **IO_DO_Set()**: **IO_E_PROT_TEMP_OVERLOAD**, **IO_E_PROT_ACTIVE** and **IO_E_PROT_FATAL**)

Returns

IO_ErrorType:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the channel id does not exist
<code>IO_E_CHANNEL_BUSY</code>	the digital output channel is currently used by another function
<code>IO_E_SW_INTERNAL</code>	Internal software error
<code>IO_E_CH_CAPABILITY</code>	The DO capability of this channel has not been activated
<code>IO_E_TASK_NO_FREE_SLOTS</code>	No more free slots to setup task function
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	the common driver init function <code>IO_Driver_Init()</code> has not been called

Remarks

For detection of open-load/short-circuit on the digital output channels `IO_DO_00..IO_DO_07`, `IO_DO_10..IO_DO_11` and `IO_DO_20..IO_DO_25` the associated ADC channel will also be configured.

For detection of overload on the digital output channels `IO_DO_20..IO_DO_25` the associated current feedback channel will also be configured.

- The following channels form groups. They have to be configured in the same mode within a group.
 - `IO_DO_00`, `IO_DO_20`
 - `IO_DO_01`, `IO_DO_21`
 - `IO_DO_02`, `IO_DO_22`
 - `IO_DO_03`, `IO_DO_23`
 - `IO_DO_04`, `IO_DO_24`
 - `IO_DO_05`, `IO_DO_25`
- Check the alternate functions of the pins used in each group. A pin can only be configured for one function at a time and it has to be the same function within the group. The alternate functions can be found at [IO_Pins.h](#)

```
IO_ErrorType IO_DO_Set ( IO_PIN      do_channel,
                        bool          do_value,
                        ubyte2 *const voltage_fb
                        )
```

Sets the state of a Digital Output.

Parameters

do_channel Digital Output:

- IO_DO_00 .. IO_DO_07
- IO_DO_10 .. IO_DO_11
- IO_DO_20 .. IO_DO_25
- IO_DO_30 .. IO_DO_35

do_value Input value:

- TRUE: High Level
- FALSE: Low Level

voltage_fb ADC value in mV of the feedback signal

- Range: 0..32780 (0mV..32780mV)

Returns

IO_ErrorType

Return values

IO_E_OK

IO_E_INVALID_CHANNEL_ID

IO_E_INVALID_DIAG_STATE

IO_E_CHANNEL_NOT_CONFIGURED

IO_E_DO_SHORT_CIRCUIT

everything fine
the channel id
does not exist
the instruction is
not permitted in
the current
diagnostic state.
the given
channel is not
configured
a short to GND
has been
detected

IO_E_DO_OPEN_LOAD	open load is detected on the output
IO_E_DO_SHORT_BATTERY	short to battery detected on the output
IO_E_DO_OPEN_LOAD_OR_SHORT_BATTERY	open load or short battery is detected on the output
IO_E_DO_DIAG_TRANSIENT_OSC	low pass of DO outputs with analog feedback is being tuned in.
IO_E_DO_CHANNEL_STARTUP	The digital output is in its startup phase
IO_E_DO_OUTPUT_DISABLED	The digital outputs are disabled by the powerstage (see IO_POWER.h for details)
IO_E_DO_OUTPUT_STARTUP_ERROR	The digital output could not be started
IO_E_PROT_USER_OVERLOAD	Output current is above the threshold configured on initialization. The output remains active.
IO_E_PROT_TEMP_OVERLOAD	Output current is between 3.0A and 4.0A (HS outputs) or 3.5A

IO_E_PROT_ACTIVE

IO_E_PROT_FATAL

and 5.5A (LS outputs) and output will be switched off if the current does not decrease to or below 3.0A (HS outputs) or to or below 3.5A (LS outputs) within the next 1 second (see [IO_Constants.h](#) for details).

Output is disabled, protection is active because of too high output current (over 4A for HS outputs or 5.5A for LS Outputs). The driver will try to re-enable the output again in 1 second (both HS and LS outputs) (see [IO_Constants.h](#) for details).

Output is disabled, protection is active because of too high output current (over 7A). The driver will try to

IO_E_PROT_REENABLE

re-enable the output again in 1 second (HS outputs with current measurement, LS Outputs) or in 10 seconds (HS outputs with overcurrent protection) (see [IO_Constants.h](#) for details).

IO_E_SW_OUTPROT_SM

The IO driver tries to re-enable the given output channel

Software malfunction (state machine error)

IO_E_ADC_INVALID

the ADC driver reported an error

IO_E_CH_CAPABILITY

The DO capability of this channel has not been activated

Remarks

- For the low-side outputs [io_do_10](#) .. [io_do_11](#) the measurement of the voltage feedback `adc_value` is only possibly in OFF-state. During ON-state the measured value will always be zero.
- If the driver is initialized as safety, the low-side outputs [io_do_10](#) .. [io_do_11](#) are allowed to be en- and disabled only while the driver is running in main state.
- For detection of open-load/short-circuit on all digital output channels the associated ADC channel will be used.

- In order to obtain a plausible value in `voltage_fb`, the `IO_DO_Set()` function has to be called a second time at least after 10 ms.

Note

- The high-side digital outputs `IO_DO_00 .. IO_DO_05` and `IO_DO_20 .. IO_DO_25` have to be enabled with `IO_POWER_Set`. Otherwise the outputs remain disabled.
- The push-pull digital outputs `IO_DO_30 .. IO_DO_35` have to be enabled with `IO_POWER_Set` as well, otherwise these outputs remain disabled.

Attention

- If the driver is initialized as non-safety, driving the low-side outputs `IO_DO_10 .. IO_DO_11` immediately after an ECU power reset can trigger a fatal output protection (`IO_E_PROT_FATAL`) due to an initial current peak.

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

[Data Structures](#) | [Typedefs](#) | [Enumerations](#) | [Functions](#)

IO_Driver.h File Reference

High level interface to IO Driver. [More...](#)

```
#include "ptypes_xe167.h" #include "IO_Constants.h"
#include "DIAG_Constants.h"
#include "IO_Pins.h"
```

Data Structures

struct [_io_driver_safety_conf](#)
Driver Safety Configuration. [More...](#)

struct [_io_driver_trap_info](#)
Contains information regarding traps/exceptions. [More...](#)

struct [_io_driver_rst_info](#)
Reset information. [More...](#)

Macros

CPU clock setting

CPU frequency in MHz

```
#define IO\_DRIVER\_SYSTEM\_CLOCK 80
```

Driver operating modes

Attention

These defines are left in for compatibility reasons. Their purpose has gone with driver version 1.2.24. Setting these parameters has no effect. Non-safety ECUs do not have a external watchdog and for safety relevant ECUs the

external watchdog is triggered by `IO_Driver_TaskBegin`. However, one of these values has to be used for the parameter `mode` of the function `IO_Driver_Init` because the parameter checks of this functions are still active (for compatibility reasons).

```
#define IO_DRIVER_MODE_DEFAULT 0x00
```

```
#define IO_DRIVER_MODE_SERVICE_WD 0x02
```

CPU reset status

```
#define IO_DRIVER_RST_STAT_NA 0x00
```

```
#define IO_DRIVER_RST_STAT_PORST 0x01
```

```
#define IO_DRIVER_RST_STAT_WD 0x02
```

```
#define IO_DRIVER_RST_STAT_WDT 0x04
```

```
#define IO_DRIVER_RST_STAT_SW 0x08
```

Options regarding the UDS security access for function reset to boot mode

```
#define IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_NONE 0x0000
```

```
#define IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_BL 0x0001
```

```
#define IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_APP 0x0002
```

Options regarding the response behavior of the bootloader after it was triggered by the application

due to an UDS reprogramming attempt.

```
#define IO_DRIVER_RTBM_UDS_RSP_NONE 0x0000
```

```
#define IO_DRIVER_RTBM_UDS_RSP_SEND 0x0001
```

Safety switch

Type of used safety switch (internal or external)

```
#define IO_DRIVER_SAFETY_SWITCH_INT 0x00
```

```
#define IO_DRIVER_SAFETY_SWITCH_EXT 0x01
```

```
#define IO_DRIVER_SAFETY_SWITCH_NONE 0x02
```

No safety switch

Not use safety switch 0,1 for the second shut-off path

```
#define IO_SAFETY_SWITCH_NONE IO_PIN_NONE
```

Typedefs

```
typedef enum  
  _io_driver_reset_reason IO_DRIVER_RESET_REASON  
  Reset reasons.
```

```
typedef struct  
  _io_driver_safety_conf IO_DRIVER_SAFETY_CONF  
  Driver Safety Configuration.
```

```
typedef struct _io_driver_trap_info IO_DRIVER_TRAP_INFO  
  Contains information regarding  
  traps/exceptions.
```

```
typedef struct _io_driver_rst_info IO_DRIVER_RESET_INFO  
  Reset information.
```

Enumerations

```
enum _io_driver_reset_reason {  
  IO_DRIVER_RESET_REASON_PORST,  
  IO_DRIVER_RESET_REASON_WDT,  
  IO_DRIVER_RESET_REASON_TRAP,  
  IO_DRIVER_RESET_REASON_SW,  
  IO_DRIVER_RESET_REASON_UNKNOWN  
}  
Reset reasons. More...
```

Functions

```
IO_ErrorType IO_Driver_Init (ubyte1 mode, const IO_DRIVER_SAFETY_CONF
```

***const safety_conf)**

Global initialization of IO driver. This function shall be called before any other driver function (except function

IO_Driver_GetVersionOfDriver() and

IO_Driver_GetResetStatus())

IO_ErrorType IO_Driver_GetMode (ubyte1 *const mode)

Returns the mode configured with **IO_Driver_Init**.

IO_ErrorType IO_Driver_GetVersionOfDriver (ubyte1 *const major, ubyte1 *const minor, ubyte2 *const patchlevel)

Returns the version number of the driver.

IO_ErrorType IO_Driver_GetVersionOfBootloader (ubyte1 *const major, ubyte1 *const minor, ubyte4 *const patchlevel)

Returns the version number of the bootloader.

IO_ErrorType IO_Driver_TaskBegin (void)

Task function for IO Driver. This function shall be called at the beginning of the task.

IO_ErrorType IO_Driver_TaskEnd (void)

Task function for IO Driver. This function shall be called at the end of the task.

ubyte2 IO_Driver_GetAutoBaudrate (void)

Returns the result of the autobaudrate detection.

IO_ErrorType IO_Driver_ResetToBootMode (ubyte2 bm_option, ubyte2 bm_option_rsp)

Resets the ECU to boot mode and instructs the bootloader to proceed with an UDS download procedure.

IO_ErrorType IO_Driver_GetResetStatus (ubyte2 *const status)

Returns the reset status.

IO_ErrorType IO_Driver_GetResetStatus_ex (IO_DRIVER_RESET_INFO *const reset_info)

Returns the reset status in more detail.

Detailed Description

High level interface to IO Driver.

The IO Driver high level interface provides a general initialization function, a version API and general task functions which shall wrap the whole user application.

Basic structure of an application

The IO Driver API provides two different types of functions:

- Initialization functions: These functions are designed to be called once at the beginning of an application.
- Task functions: These functions are designed to be called periodically at runtime.

The function **IO_Driver_Init()** needs to be the first function, called during the initialization.

All safety critical IO initialization functions need to be called before the first call of **IO_Driver_TaskBegin()**.

All task functions need to be enclosed by the functions **IO_Driver_TaskBegin()** and **IO_Driver_TaskEnd()**

Example of an application:

```
void task (void)
{
    IO_Driver_TaskBegin();

    // User Application
    // and calls to driver task functions.

    IO_Driver_TaskEnd();
}

void main (void)
{
    ubyte4 timestamp;

    //-----//
    // start of driver initialization //
    //-----//

    // IO_Driver_Init() is the first function:
    IO_Driver_Init( IO_DRIVER_MODE_SERVICE_WD, NULL );      //
    // configure automatic WD servicing

    //-----//
    // end of driver initialization //
    //-----//

    //-----//
    // start of safety critical IO initialization //
    //-----//
```

```

// initialize all the safety critical IO functions
//-----//
// end of safety critical IO initialization //
//-----//

//-----//
// from now on only task functions are called //
//-----//
while (1){
    IO_RTC_StartTime(&timestamp);

    task();

    while (IO_RTC_GetTimeUS(timestamp) < 5000);
}
}

```

The `task` function is called every 5000us = 5ms. Please refer to the [Real Time Clock](#) documentation for details on how to use the RTC functions.

Macro Definition Documentation

#define IO_DRIVER_MODE_DEFAULT 0x00

Default mode for the IO-Driver with:

- need to manually serve the Window Watchdog ([IO_WD.h](#))

#define IO_DRIVER_MODE_SERVICE_WD 0x02

Bit to set IO-Driver mode to service the Window Watchdog automatically. The Watchdog will be served automatically within function [IO_Driver_TaskBegin](#). Due to a Watchdog window of 2ms..45ms (see [IO_WD.h](#)), in this mode the application cycle time must not exceed 45ms.

#define IO_DRIVER_RST_STAT_NA 0x00

Reset status could not be determined

#define IO_DRIVER_RST_STAT_PORST 0x01

Reset status for a Poweron Reset

#define IO_DRIVER_RST_STAT_SW 0x08

Reset status for a Software reset

#define IO_DRIVER_RST_STAT_WD 0x02

Reset status for a Window Watchdog reset

#define IO_DRIVER_RST_STAT_WDT 0x04

Reset status for a Watchdog Timer reset

#define IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_APP 0x0002

Reset to boot mode. Authentication has been done already by the application software. Not recommended!

#define IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_BL 0x0001

Reset to boot mode. Authentication will be done in the bootloader. This is the recommended option

#define IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_NONE 0x0000

No reprogramming attempt. Bootloader will not stop during boot.

#define IO_DRIVER_RTBM_UDS_RSP_NONE 0x0000

The bootloader will not send any response without receiving the request for reprogramming by itself.

#define IO_DRIVER_RTBM_UDS_RSP_SEND 0x0001

The bootloader will send the response to the request for reprogramming that has been received by the application.

#define IO_DRIVER_SAFETY_SWITCH_EXT 0x01

External safety switch

#define IO_DRIVER_SAFETY_SWITCH_INT 0x00

Internal safety switch

#define IO_DRIVER_SAFETY_SWITCH_NONE 0x02

No safety switch

#define IO_DRIVER_SYSTEM_CLOCK 80

CPU frequency in MHz

#define IO_SAFETY_SWITCH_NONE IO_PIN_NONE

Pin NONE, PWM safety switch none configuration

Typedef Documentation

typedef enum `_io_driver_reset_reason` `IO_DRIVER_RESET_REASON`

Reset reasons.

Remarks

- Depending on the ECU not all or additional reasons are available.
- Definitions may be used instead of enumerations.

typedef struct `_io_driver_safety_conf` `IO_DRIVER_SAFETY_CONF`

Driver Safety Configuration.

This structure is used to pass the configuration for a safety critical application to the IO-Driver.

Enumeration Type Documentation

enum `_io_driver_reset_reason`

Reset reasons.

Remarks

- Depending on the ECU not all or additional reasons are available.
- Definitions may be used instead of enumerations.

Enumerator:

<i>IO_DRIVER_RESET_REASON_PORST</i>	Power-On reset
<i>IO_DRIVER_RESET_REASON_WDT</i>	Watchdog-Timer reset
<i>IO_DRIVER_RESET_REASON_TRAP</i>	Reset due to exception
<i>IO_DRIVER_RESET_REASON_SW</i>	Reset due to call of <code>IO_Driver_ResetToBootMode</code> or <code>IO_Driver_Reset</code>
<i>IO_DRIVER_RESET_REASON_UNKNOWN</i>	Unknown reset

Function Documentation

ubyte2 IO_Driver_GetAutoBaudrate (void)

Returns the result of the autobaudrate detection.

This function returns the result of the autobaudrate detection which was executed during startup (within the bootloader).

Returns

ubyte2 autoBaudrate

Return values

0 the autoBaudrate detection could not verify a valid baudrate

125 a baudrate of 125kbps was detected

250 a baudrate of 250kbps was detected

500 a baudrate of 500kbps was detected

IO_ErrorType IO_Driver_GetMode (**ubyte1** *const **mode**)

Returns the mode configured with **IO_Driver_Init**.

This function returns the mode passed when initializing the IO-Driver with **IO_Driver_Init**.

Parameters

mode Returned mode (see **IO_Driver_Init** for details)

Returns

IO_ErrorType

Return values

IO_E_OK everything fine

IO_E_DRIVER_NOT_INITIALIZED the common driver init function has not been called before

Attention

This function has been left in for compatibility reasons. Its functionality has been removed with driver version 1.2.24. The returned value for `mode` is always **IO_DRIVER_MODE_DEFAULT**

IO_ErrorType IO_Driver_GetResetStatus (ubyte2 *const status)

Returns the reset status.

This function returns the status/reason for the last CPU-reset.

Parameters

status Returned status. One of:

- **IO_DRIVER_RST_STAT_NA**
- **IO_DRIVER_RST_STAT_PORST**
- **IO_DRIVER_RST_STAT_WD**
- **IO_DRIVER_RST_STAT_WDT**
- **IO_DRIVER_RST_STAT_SW**

Returns

IO_ErrorType

Return values

IO_E_OK everything fine
IO_E_NULL_POINTER null pointer has been passed

IO_ErrorType

IO_Driver_GetResetStatus_ex (IO_DRIVER_RESET_INFO *const reset_info)

Returns the reset status in more detail.

Parameters

[out] **reset_info** Information about reset

Returns

IO_ErrorType

Return values

IO_E_OK Everything ok.
IO_E_NULL_POINTER A null pointer has been passed
IO_E_RESET_COUNTER_INVALID The reset counter is invalid. No reset or trap info is available (This indicates a SW or HW malfunction).
IO_E_SBRAM_CONTENT_INVALID The trap information stored in the SBRAM is invalid. No trap info is

available (This indicates a SW or HW malfunction).

```
IO_ErrorType IO_Driver_GetVersionOfBootloader ( ubyte1 *const major,  
                                                ubyte1 *const minor,  
                                                ubyte4 *const patchlevel  
                                                )
```

Returns the version number of the bootloader.

Parameters

major Major version
minor Minor version
patchlevel Patchlevel

Returns

IO_ErrorType

Return values

IO_E_OK everything fine
IO_E_NULL_POINTER null pointer has been passed

```
IO_ErrorType IO_Driver_GetVersionOfDriver ( ubyte1 *const major,  
                                             ubyte1 *const minor,  
                                             ubyte2 *const patchlevel  
                                             )
```

Returns the version number of the driver.

Parameters

major Major version
minor Minor version
patchlevel Patchlevel

Returns

IO_ErrorType

Return values

IO_E_OK everything fine
IO_E_NULL_POINTER null pointer has been passed

```

IO_ErrorType
IO_Driver_Init      ( ubyte1                                mode,
                     const IO_DRIVER_SAFETY_CONF *const safety_conf
                     )

```

Global initialization of IO driver. This function shall be called before any other driver function (except function **IO_Driver_GetVersionOfDriver()** and **IO_Driver_GetResetStatus()**)

- Initializes SPI Devices
- Switches off all power outputs
- Initializes the RTC
- Switches on the interrupts of the CPU
- Initializes the measurement of UBAT
- Initializes and configures the Window Watchdog

Parameters

- mode** A bitfield that allows to configure various settings of the IO-Driver. One or more of:
- **IO_DRIVER_MODE_DEFAULT**: Default mode with manual triggering.
 - **IO_DRIVER_MODE_SERVICE_WD**: The Watchdog is serviced automatically by the IO-Driver.
- safety_conf** Safety Configuration if the hardware is supporting and used as safety device. Otherwise set this parameter to NULL.

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_WRONG_HW_TYPE	wrong HW-Type detected
IO_E_BUSY	driver is just initialized
IO_E_SAFETY_NOT_SUPPORTED	the HW does not support safety features
IO_E_INVALID_SAFETY_CONFIG	the safety configuration is faulty
IO_E_INVALID_CRC	the CRC from the APDB is invalid
IO_E_TASK_NO_FREE_SLOTS	No more free slots to setup task function

Remarks

Initializing the driver with a valid safety configuration might take about 65ms.

Attention

The parameter mode is left in for compatibility reasons. Its functionality has been removed with driver version 1.2.24. Setting this parameter has no effect. Non-safety ECU do not have a external watchdog and for safety relevant ECUs the external watchdog is triggered by `IO_Driver_TaskBegin`.

```
IO_ErrorType IO_Driver_ResetToBootMode ( ubyte2 bm_option,  
                                           ubyte2 bm_option_rsp  
                                           )
```

Resets the ECU to boot mode and instructs the bootloader to proceed with an UDS download procedure.

Parameters

bm_option

Option for Bootloader

- **IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_BL**: Reset to boot mode. Authentication will be handled by the bootloader. This is considered the standard compliant way and thus recommended.
- **IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_APP**: Reset to boot mode. Authentication will be handled by the application software. It is the software integrator's responsibility to secure the ECU from unauthorized access in this case. This option is considered to be not compliant to the standard and thus NOT RECOMMENDED! It is only possible to unlock security access level 1 via this function (reprogramming of application).

bm_option_rsp Defines if the response to either the UDS request to switch to the programming session or the security access has to be sent by the bootloader or not.

- **IO_DRIVER_RTBM_UDS_RSP_NONE**: The bootloader will not send any response without receiving the request for reprogramming by itself.
- **IO_DRIVER_RTBM_UDS_RSP_SEND**: Instruct the bootloader to send a response to the request for reprogramming that has been received by the application. The response depends on the parameter `bm_option`: if called with `IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_BL`, the application responds to the request to switch to the programming session (UDS service 0x10 response message <06 50 02 00 32 01 F4 55>); if called with `IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_APP`,

the application responds to "send key" (UDS service 0x27 response message <02 67 xx 55 55 55 55 55>, where "xx" is the respective sub-function selected in the branding block, or the default 0x62).

Returns

IO_ErrorType

Return values

IO_E_OK

Function executed Successfully (This error will never be shown because if the function executed successfully a CPU reset has been done and the function will never return).

IO_E_CHANNEL_NOT_CONFIGURED

The I/O driver init function has not been called before.

IO_E_INVALID_PARAMETER

An invalid parameter has been passed.

Remarks

- Prior to calling this function the pre-programming steps according to ISO 14229-1:2013 have to be handled by the application software.
- After receiving a request to switch to the programming session the application software shall call this function to switch to the programming session if the machine conditions allow it.
- It is recommended that after the application received a request to switch to the programming session (CAN message <02 10 02>) the function `IO_Driver_ResetToBootMode(IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_BL)` is be called. If the application needs time to shut down then a RCRRP (service 0x78) shall be sent to the diagnose tester.
- If this function is called with parameter **IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_BL** the bootloader will send the positive response to the request to switch to the programming session (CAN message <06 50 02 00 32 01 F4>).
- If this function is called with parameter **IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_APP** the application software must add the bootloaders P2 timing parameters into the response of request to switch to the programming session.
- Be aware that the option **IO_DRIVER_RTBM_UDS_RSP_NONE** for parameter `bm_option_rsp` is not recommended. If the application sends the response to the request to switch to the programming session before it calls the function `IO_Driver_ResetToBootMode`, then there will be a gap where the ECU can not answer any requests from the tester until the bootloader including its UDS server is started. For the TTC30X type of ECUs this time can be up to 30ms (time from calling the function

IO_Driver_ResetToBootMode until the UDS server of the bootloader is ready to accept commands).

Attention

Be aware that resetting to boot mode during normal operation is dangerous and in general NOT ALLOWED. The application software is only authorized to call this function if:

- The vehicle is inactive/not moving
- The engine is not running
- the risk of any other hazard by stopping the application software at the time of the reprogramming request can be ruled out

IO_ErrorType IO_Driver_TaskBegin (void)

Task function for IO Driver. This function shall be called at the beginning of the task.

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_NOT_CONFIGURED	Driver was not initialized correctly
IO_E_UART_PARITY	parity check failed
IO_E_UART_OVERFLOW	HW receive buffer overrun
IO_E_UART_BUFFER_FULL	SW receive queue is full and data has been lost

IO_ErrorType IO_Driver_TaskEnd (void)

Task function for IO Driver. This function shall be called at the end of the task.

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_NOT_CONFIGURED	Driver was not initialized correctly
IO_E_UART_PARITY	parity check failed
IO_E_UART_OVERFLOW	HW receive buffer overrun
IO_E_UART_BUFFER_FULL	SW receive queue is full and data has been lost

Main Page		Related Pages	Data Structures	Files
File List	Globals			
inc				

Functions

IO_EEPROM.h File Reference

IO Driver functions for EEPROM. [More...](#)

```
#include "IO_Driver.h"
```

Functions

IO_ErrorType **IO_EEPROM_Init** (void)
Initialization of the EEPROM driver.

IO_ErrorType **IO_EEPROM_DeInit** (void)
Deinitializes the EEPROM driver.

IO_ErrorType **IO_EEPROM_Read** (ubyte2 offset, ubyte2 length, bool use_crc, ubyte1 *const data)
Read data from the EEPROM.

IO_ErrorType **IO_EEPROM_Write** (ubyte2 offset, ubyte2 length, bool use_crc, const ubyte1 *const data)
Write data to the EEPROM.

IO_ErrorType **IO_EEPROM_GetStatus** (void)

Returns the status of the EEPROM driver.

Detailed Description

IO Driver functions for EEPROM.

The EEPROM driver functions allow writing to the SPI EEPROM as well as reading from it.

The EEPROM driver is a high level SPI device driver and uses the low level SPI driver.

The memory amounts 8128 Byte, because 64 Byte of the EEPROM are reserved for internal use.

Since the content of the EEPROM is not initialized or verified during the production procedure, there is no guarantee of a defined initial content (of 0xFF) on any position.

The communication with the EEPROM is handled in a cyclic manner. EEPROM operations take plenty of time. A write as well as a read process is only triggered by calling the respective function. A status function is provided which returns whether the driver has finished the last task or not.

The EEPROM driver gives the option of calculating CRC32 checksums (size of 4 Bytes, polynome 0x04C11DB7 and initialization value 0x00000000) on write and read operations. When enabled, the user must take care to leave enough space for the checksum and not to overwrite it. For example, writing to offset zero data with a size of 6 Bytes, the checksum will be appended to this 6 Bytes. This means, that offsets 0-5 contain the actual written data and offsets 6-9 contain the checksum. The user must not overwrite the bytes 6-9 via other calls of **IO_EEPROM_Write**. Otherwise the next read operation with checksum enabled will result in an **IO_E_EEPROM_CRC_MISMATCH** error.

EEPROM code examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

EEPROM initialization example:

EEPROM does not need an explicit initialization. The EEPROM driver is initialized by **IO_Driver_Init()**.

EEPROM write example:

```
ubyte1 data[6] = {0, 1, 2, 3, 4, 5};

// check if EEPROM is busy
if (IO_EEPROM_GetStatus() == IO_E_OK)
{
    // if not busy write data without checksum
    IO_EEPROM_Write(0, 6, FALSE, data);
}

// write is complete when IO_EEPROM_GetStatus()
// returns IO_E_OK again.
```

EEPROM read example:

```
ubyte1 data[2000] = {0};

// check if EEPROM is busy
if (IO_EEPROM_GetStatus() == IO_E_OK)
{
    // if not busy start reading without
    // checksum
    IO_EEPROM_Read(0, 2000, FALSE, data);
}

// data is not yet available!!
```

```
// data is available when IO_EEPROM_GetStatus()  
    returns IO_E_OK again.
```

Function Documentation

IO_ErrorType IO_EEPROM_DeInit (void)

Deinitializes the EEPROM driver.

Deinitializes the module. Allows re-initialization by

`IO_EEPROM_Init()`

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_NOT_CONFIGURED the module is not initialized

IO_ErrorType IO_EEPROM_GetStatus (void)

Returns the status of the EEPROM driver.

Returns whether the EEPROM is idle or if a read or write operation is ongoing.

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine,
driver is idle

IO_E_EEPROM_CRC_MISMATCH the stored and

IO_E_BUSY

calculated CRC value of a read operation do not match

a read or a write operation is ongoing, driver is busy.

IO_E_CHANNEL_NOT_CONFIGURED

the module is not initialized

IO_ErrorType IO_EEPROM_Init (void)

Initialization of the EEPROM driver.

Initialization of EEPROM driver.

- Initializes internal data structure
- Configures SPI driver

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_BUSY

module has been initialized before

Remarks

- Module is initialized only once. To re-initialize the module, the function **IO_EEPROM_DeInit()** needs to be called.
- The EEPROM driver is initialized when the **IO_Driver_Init()** function is called. Therefore it will

return `IO_E_CHANNEL_BUSY` if it is called after this function. This means that `IO_EEPROM_Init()` needs to be called only when `IO_Driver_Init()` is not used in the respective application.

```
IO_ErrorType IO_EEPROM_Read ( ubyte2          offset,  
                                ubyte2          length,  
                                bool           use_crc,  
                                ubyte1 *const data  
                                )
```

Read data from the EEPROM.

The function only triggers a read operation. The read operation can take several cycles to be completed, depending on the SPI load and the amount of data to be read.

The read data is available on the address where the data parameter points to as soon as the read operation is finished. The state can be polled using the `IO_EEPROM_GetStatus()` function.

Parameters

offset EEPROM memory offset (0..8127)
length Length of data to be read (0..8128)
use_crc Indicates if the CRC value stored by
 `IO_EEPROM_Write` should be read and
 evaluated
data Pointer to data

Returns

`IO_ErrorType`

Return values

IO_E_OK	everything fine
IO_E_BUSY	EEPROM module is still busy
IO_E_EEPROM_RANGE	invalid address offset or range
IO_E_NULL_POINTER	a null pointer has been passed
IO_E_CHANNEL_NOT_CONFIGURED	the module is not initialized

IO_ErrorType

```

IO_EEPROM_Write      ( ubyte2      offset,
                      ubyte2      length,
                      bool         use_crc,
                      const ubyte1 *const data
                      )

```

Write data to the EEPROM.

The function triggers a write operation. The write operation can take several cycles to be completed, depending on the SPI load and the amount of data to be read.

The write operation is completed as soon as the SPI communication has been finished. The state can be polled using the **IO_EEPROM_GetStatus()** function.

Parameters

- offset** EEPROM memory offset (0..8127)
- length** Length of data to be written (0..8128)
- use_crc** Indicates if a CRC value (4-bytes) should be stored automatically immediately after the written

bytes. If TRUE, the maximum writable address is 8124.

data Pointer to data

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_BUSY

EEPROM module
is still busy

IO_E_EEPROM_RANGE

invalid address
offset or range

IO_E_NULL_POINTER

a null pointer has
been passed

IO_E_CHANNEL_NOT_CONFIGURED

the module is not
initialized

Note

Please note that the parameter 'data' must not be a local variable since the driver does not create a copy of the content but saves the address of the pointer.



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page		Related Pages	Data Structures	Files
File List	Globals			
inc				

Functions

IO_EEPROM_Preload.h File Reference

Pre-load functions for EEPROM. [More...](#)

```
#include "IO_Driver.h"
```

Functions

IO_ErrorType **IO_EEPROM_PreloadInit** (void)
Preload initialization of the EEPROM driver.

IO_ErrorType **IO_EEPROM_PreloadDeInit** (void)
De-initializes the preloaded EEPROM driver.

IO_ErrorType **IO_EEPROM_PreloadRead** (**ubyte2** offset, **ubyte2** length, **bool** use_crc, **ubyte1** *const data)
Read data from the pre-loaded EEPROM module.

IO_ErrorType **IO_EEPROM_PreloadWrite** (**ubyte2** offset, **ubyte2** length, **bool** use_crc, const **ubyte1** *const data)

Write data to the pre-loaded EEPROM module.

IO_ErrorType IO_EEPROM_PreloadStatus (void)
Returns the status of the pre-load EEPROM driver.

IO_ErrorType IO_EEPROM_PreloadTask (void)
Task function for the pre-load EEPROM driver.

Detailed Description

Pre-load functions for EEPROM.

The EEPROM pre-load functions allow writing to the SPI EEPROM as well as reading from it without initializing the whole IO driver. Therefore it is possible to read data from the memory which could be necessary for a proper initialization of the IO driver (e.g. safety configuration data).

The EEPROM driver is a high level SPI device driver and uses the low level SPI driver.

The memory amounts 8128 Byte, because 64 Byte of the EEPROM are reserved for internal use.

Since the content of the EEPROM is not initialized or verified during the production procedure, there is no guarantee of a defined initial content (of 0xFF) on any position.

The communication with the EEPROM is handled in a cyclic manner. EEPROM operations take plenty of time. A write as well as a read process is only triggered by calling the respective function. A status function is provided which returns whether the driver has finished the last task or not.

The EEPROM driver gives the option of calculating CRC32 checksums (size of 4 Bytes, polynome 0x04C11DB7 and initialization value 0x00000000) on write and read operations. When enabled, the user must take care to leave enough space for the checksum and not to overwrite it. For example, writing to offset zero data with a size of 6 Bytes, the checksum will be appended to this 6 Bytes. This means, that offsets 0-5 contain the actual written data and offsets 6-9 contain the checksum. The user must not

overwrite the bytes 6-9 via other calls of **IO_EEPROM_Write**. Otherwise the next read operation with checksum enabled will result in an **IO_E_EEPROM_CRC_MISMATCH** error.

EEPROM pre-load code examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

Typical usage of the pre-load functions:

```
ubyte1      data[50] = {0};

IO_EEPROM_PreloadInit();

IO_EEPROM_PreloadRead(0, 50, FALSE, data);
while( IO_EEPROM_PreloadStatus() != IO_E_OK)
{
    IO_EEPROM_PreloadTask();
}

IO_EEPROM_PreloadDeInit();
```

EEPROM write example:

```
ubyte1 data[6] = {0, 1, 2, 3, 4, 5};

// check if EEPROM is busy
if (IO_EEPROM_PreloadStatus() == IO_E_OK)
{
    // if not busy write data without checksum
    IO_EEPROM_PreloadWrite(0, 6, FALSE, data);

    // wait until write cycle is over
    while( IO_EEPROM_PreloadStatus() !=
        IO_E_OK)
    {
        IO_EEPROM_PreloadTask();
    }
}
```

EEPROM read example:

```
ubyte1 data[2000] = {0};

// check if EEPROM is busy
if (IO_EEPROM_PreloadStatus() == IO_E_OK)
{
    // if not busy start reading without
    // checksum
    IO_EEPROM_PreloadRead(0, 2000, FALSE,
        data);

    // wait until read cycle is over
    while( IO_EEPROM_PreloadStatus() !=
        IO_E_OK)
    {
        IO_EEPROM_PreloadTask();
    }
}
```

Note

It is highly recommended to de-initialize the EEPROM pre-load module before calling the **IO_Driver_Init()** function.

Function Documentation

IO_ErrorType IO_EEPROM_PreloadDeInit (void)

De-initializes the preloaded EEPROM driver.

De-initializes the EEPROM module as well as the SPI driver.

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_NOT_CONFIGURED the module is not initialized

IO_ErrorType IO_EEPROM_PreloadInit (void)

Preload initialization of the EEPROM driver.

Initialization of EEPROM driver.

- Initializes internal data structure
- Configures SPI driver

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_BUSY module has been initialized

before

Remarks

- Module is initialized only once. To re-initialize the module, the function `IO_EEPROM_PreloadDeInit()` needs to be called.

IO_ErrorType

```
IO_EEPROM_PreloadRead      ( ubyte2      offset,  
                             ubyte2      length,  
                             bool         use_crc,  
                             ubyte1 *const data  
                             )
```

Read data from the pre-loaded EEPROM module.

The function only triggers a read operation. The read operation can take several cycles to be completed, depending on the SPI load and the amount of data to be read.

The read data is available on the address where the data parameter points to as soon as the read operation is finished. The state can be polled using the `IO_EEPROM_PreloadStatus()` function.

Parameters

- offset** EEPROM memory offset (0..8127)
length Length of data to be read (0..8128)
use_crc Indicates if the CRC value stored by `IO_EEPROM_Write` should be read and evaluated
data Pointer to data

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_BUSY

EEPROM module
is still busy

IO_E_EEPROM_RANGE

invalid address
offset or range

IO_E_NULL_POINTER

a null pointer has
been passed

IO_E_CHANNEL_NOT_CONFIGURED the module is not
initialized

IO_ErrorType IO_EEPROM_PreloadStatus (void)

Returns the status of the pre-load EEPROM driver.

Returns whether the EEPROM is idle or if a read or write operation is ongoing.

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine,
driver is idle

IO_E_EEPROM_CRC_MISMATCH

the stored and
calculated CRC
value of a read
operation do not
match

IO_E_BUSY

a read or a write
operation is

ongoing, driver is busy.

IO_E_CHANNEL_NOT_CONFIGURED the module is not initialized

IO_ErrorType IO_EEPROM_PreloadTask (void)

Task function for the pre-load EEPROM driver.

Handles the SPI communication and calls the EEPROM read/write function task.

When using the EEPROM pre-load functions it is mandatory to call the task function periodically. Otherwise no SPI communication takes place and any read or write operation will fail.

Returns

IO_ErrorType

Return values

IO_E_OK everything fine

IO_E_BUSY SPI communication is still ongoing

IO_ErrorType

IO_EEPROM_PreloadWrite (**ubyte2** **offset,**
ubyte2 **length,**
bool **use_crc,**
const ubyte1 *const data
)

Write data to the pre-loaded EEPROM module.

The function triggers a write operation. The write operation can take several cycles to be completed, depending on the SPI load and the amount of data to be read.

The write operation is completed as soon as the SPI communication has been finished. The state can be polled using the **IO_EEPROM_PreloadStatus()** function.

Parameters

- offset** EEPROM memory offset (0..8127)
- length** Length of data to be written (0..8128)
- use_crc** Indicates if a CRC value (4-bytes) should be stored automatically immediately after the written bytes. If TRUE, the maximum writable address is 8124.
- data** Pointer to data

Returns

IO_ErrorType

Return values

- | | |
|------------------------------------|---------------------------------|
| IO_E_OK | everything fine |
| IO_E_BUSY | EEPROM module is still busy |
| IO_E_EEPROM_RANGE | invalid address offset or range |
| IO_E_NULL_POINTER | a null pointer has been passed |
| IO_E_CHANNEL_NOT_CONFIGURED | the module is not initialized |

Note

Please note that the parameter 'data' must not be a local variable since the driver does not create a copy of the content but saves the address of the pointer.



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

Functions

IO_LED.h File Reference

IO driver functions for LED. [More...](#)

```
#include "IO_Driver.h"
```

Functions

IO_ErrorType **IO_LED_ChannelInit** (**IO_PIN** led_channel)
Setup one LED channel.

IO_ErrorType **IO_LED_ChannelDeInit** (**IO_PIN** led_channel)
Deinitializes one LED channel.

IO_ErrorType **IO_LED_Set** (**IO_PIN** led_channel, **bool** led_value, **ubyte2** *measurement)
Sets the state of a LED channel.

Detailed Description

IO driver functions for LED.

The LED module allows to turn on and off LEDs on the designated channels `IO_LED_00` ... `IO_LED_07`.

LED Code Examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

Example for a LED initialization

```
IO_LED_ChannelInit( IO_LED_00 );  
IO_LED_ChannelInit( IO_LED_01 );
```

Example to turn on/off a LED

```
ubyte2    led0_fb;  
ubyte2    led1_fb;  
  
IO_LED_Set( IO_LED_00, TRUE, &led0_fb );  
           // turns LED on  
IO_LED_Set( IO_LED_01, FALSE, &led1_fb );  
           // turns LED off
```

Function Documentation

IO_ErrorType IO_LED_ChannelDeInit (IO_PIN led_channel)

Deinitializes one LED channel.

deinitializes the given LED channel, allows re-initialization by `IO_LED_ChannelInit()`

Parameters

led_channel LED channel, one of:

- `IO_LED_00 ... IO_LED_07`

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_INVALID_CHANNEL_ID

the given channel id does not exist

IO_E_CHANNEL_NOT_CONFIGURED the given channel is not configured

Remarks

- The following channels form groups. They have to be configured in the same mode within a group.
 - `IO_LED_00 .. IO_LED_01`
 - `IO_LED_02 .. IO_LED_03`
 - `IO_LED_04 .. IO_LED_05`
 - `IO_LED_06 .. IO_LED_07`

IO_ErrorType IO_LED_Channellnit (IO_PIN led_channel)

Setup one LED channel.

Parameters

led_channel LED channel, one of:

- `IO_LED_00 ... IO_LED_07`

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything fine

`IO_E_INVALID_CHANNEL_ID`

the channel id does not exist

`IO_E_CHANNEL_BUSY`

the ADC input channel is currently used by another function

`IO_E_DRIVER_NOT_INITIALIZED` The common driver init function has not been called before

`IO_E_CH_CAPABILITY`

The LED capability of this channel has not been activated

Remarks

- The following channels form groups. They have to be configured in the same mode within a group.
 - `IO_LED_00 .. IO_LED_01`
 - `IO_LED_02 .. IO_LED_03`
 - `IO_LED_04 .. IO_LED_05`
 - `IO_LED_06 .. IO_LED_07`
- Check the alternate functions of the pins used in each group. A pin can only be configured for one function at

a time and it has to be the same function within the group. The alternate functions can be found at [IO_Pins.h](#)

```
IO_ErrorType IO_LED_Set ( IO_PIN  led_channel,  
                          bool     led_value,  
                          ubyte2 * measurement  
                          )
```

Sets the state of a LED channel.

Parameters

- led_channel** LED channel, one of:
- [IO_LED_00](#) ... [IO_LED_07](#)
- led_value** Input value:
- `TRUE`: Turns LED on
 - `FALSE`: Turns LED off
- measurement** If the LED was turned on (`led_value = TRUE`) this parameter returns the actual current value (Range: 0...27.600mA) If the LED was turned off (`led_value = FALSE`) this parameter returns the actual voltage on the pin in mV (Range: 0...10.500V)

Returns

[IO_ErrorType](#)

Return values

[IO_E_OK](#)

[IO_E_NULL_POINTER](#)

[IO_E_INVALID_CHANNEL_ID](#)

everything fine
A NULL pointer
has been passed
the channel id
does not exist

IO_E_BUSY	the channel is not settled so far (after switching a LED)
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_ADC_CHANNEL_STARTUP	Channel is in initialization phase
IO_E_FET_PROTECTION	an overcurrent was detected
IO_E_ADC_INVALID	The ADC value is invalid/not available
IO_E_INVALID_PARAMETER	invalid parameter has been passed

Main Page	Related Pages	Data Structures	Files	
File List	Globals			
inc				
IO_NodeID.h File Reference				Functions

IO Driver functions for reading the NodeID pins. [More...](#)

```
#include "IO_Driver.h"
```

Functions

IO_ErrorType **IO_NodeID_GetModifierStartup** (**ubyte1** *const modifier, **ubyte1** *const nodeid)
Retrieves the Modifier and the NodeID used on ECU startup by the bootloader.

IO_ErrorType **IO_NodeID_GetModifier** (**ubyte2** ubat, **ubyte2** nodeid_0, **ubyte2** nodeid_1, **ubyte1** *const modifier)
Calculates the modifier for the given voltage levels.

Detailed Description

IO Driver functions for reading the NodeID pins.

Attention

NodeID pin functionality is NOT provided for HY-TTC32H or HY-TTC32SH, since those pins are used for the second CAN interface.

The HY-TTC 30 has two connector pins **IO_ADC_NODE_ID_0** and **IO_ADC_NODE_ID_1** which allow two change the NodeID of the ECU.

A NodeID pin can be connected to either battery voltage, sensor supply, ground or left floating.

The final NodeID used by the bootloader and/or the application, is calculated with

$$\text{NodeID} = \text{BaseID} + \text{Modifier}$$

where `BaseID` is the value stored in the field `NodeNumber` of the APDB (see [ApdbType](#)) and `Modifier` the value read from the NodeID pins according to the following table for non-safety related devices:

Modifier	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
NODEID_0	Floating	Ground	SS	UBAT+	Ground	SS	UBAT+	Ground	SS	UBAT+
NODEID_1	Floating	Ground	Ground	Ground	SS	SS	SS	UBAT+	UBAT+	UBAT+

For safety-related devices the following table applies:

Modifier	+0	+3	+5	+7
NODEID_0	Floating	UBAT+	SS	Ground
NODEID_1	Floating	Ground	SS	UBAT+

All combinations not mentioned in the above table are an invalid state. The `Modifier` determined by the bootloader is saved to the EEPROM automatically.

The `Modifier` and actual NodeID determined by the bootloader can be retrieved using the function [IO_NodeID_GetModifierStartup](#). The current state of the NodeID pins can be retrieve during the function [IO_NodeID_GetModifier](#).

NodeID code examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

Startup NodeID initialization example:

Reading the NodeID at startup does not need initialization.

Startup NodeID read example:

```
ubytel modifier;
ubytel nodeid;

IO_NodeID_GetModifierStartup( &modifier    // will contain the NodeID
    modifier (0..9)
                                , &nodeid); // will contain the used
    NodeID (0..127)
```

Current NodeID initialization example:

```
// The range parameter is ignored for these inputs
IO_ADC_ChannelInit( IO_ADC_UBAT, IO_ADC_ABSOLUTE, 0, NULL );
    // Initialize analog channel of UBAT
IO_ADC_ChannelInit( IO_ADC_NODE_ID_0, IO_ADC_ABSOLUTE, 0, NULL );
    // Initialize analog channel of NODEID_0
IO_ADC_ChannelInit( IO_ADC_NODE_ID_1, IO_ADC_ABSOLUTE, 0, NULL );
    // Initialize analog channel of NODEID 1
```

Current NodeID read example:

[illegible]

Function Documentation

```
IO_ErrorType IO_NodeID_GetModifier ( ubyte2      ubat,  
                                     ubyte2      nodeid_0,  
                                     ubyte2      nodeid_1,  
                                     ubyte1 *const modifier  
                                     )
```

Calculates the modifier for the given voltage levels.

Parameters

ubat Battery voltage (**IO_ADC_UBAT**) in mV
nodeid_0 Voltage of pin **IO_ADC_NODE_ID_0** [mV]
nodeid_1 Voltage of pin **IO_ADC_NODE_ID_1** [mV]
modifier Calculated modifier (Range: 0..9)

Returns

IO_ErrorType

Return values

IO_E_OK everything fine
IO_E_NODEID_PINS_INVALID the given voltage levels represent no valid Modifier
IO_E_NULL_POINTER a null pointer has been passed

Remarks

It is recommended to check the validity of the sensor supply voltage before calling the **IO_NodeID_GetModifier()** function. Otherwise it is possible that an invalid value is computed (e.g. if the sensor supply is short-circuited to GND).

```
IO_ErrorType IO_NodeID_GetModifierStartup ( ubyte1 *const modifier,  
                                             ubyte1 *const nodeid  
                                             )
```

Retrieves the Modifier and the NodeID used on ECU startup by the bootloader.

Parameters

modifier returned NodeID Modifier (Range: 0..9)
nodeid returned NodeID (Range: 0..127 for non-safety variants, 0..64 for safety-variants)

Returns

IO_ErrorType

Return values

IO_E_OK everything fine
IO_E_NODEID_PINS_INVALID content of EEPROM and pins invalid (no valid

	modifier could be determined)
IO_E_NODEID_EEPROM_FALLBACK	Value of NodeID pins was invalid but EEPROM values could be used as Modifier
IO_E_NODEID_EEPROM_INVALID	Modifier stored in the EEPROM was not valid and rewritten with the current Modifier read from NodeID pins
IO_E_NODEID_EEPROM_MISMATCH	Both modifier gotten from pins and EEPROM are valid, but did not match on startup (values from the NodeID pins have been used)
IO_E_SW_INTERNAL	an internal error has occurred
IO_E_NULL_POINTER	a NULL pointer has been passed to the function

Remarks

The following five combinations are possible for the calculation of the `modifier` by the bootloader:

- Both, the current value of the NodeID pins and the values stored in the EEPROM are valid and match:
The function returns **IO_E_OK**, the value of the parameter `modifier` corresponds to the modifier determined via the NodeID pins (and matches the EEPROM values).
- Both, the current value of the NodeID pins and the values stored in the EEPROM are valid but do not match (because, e.g., wiring has changed):
The function returns **IO_E_NODEID_EEPROM_MISMATCH**, the value of the parameter `modifier` corresponds to the modifier determined via the NodeID pins. Before starting the application, the bootloader has updated the EEPROM values with the current state of the pins such that upon the next start-up the function returns **IO_E_OK** if the pin values do not change.
- The value of the NodeID pins is valid but the values stored in the EEPROM are invalid (e.g. first start-up of device or after the EEPROM has been cleared):
The function returns **IO_E_NODEID_EEPROM_INVALID**, the value of the parameter `modifier` corresponds to the modifier determined via the NodeID pins. Before starting the application, the bootloader has updated the EEPROM values with the current state of the pins such that upon the next start-up the function returns **IO_E_OK** if the pin values do not change.
- The value of the NodeID pins is invalid and the values stored in the EEPROM are valid:
The bootloader will use the values in the EEPROM to calculate the modifier, the function returns **IO_E_NODEID_EEPROM_FALLBACK**. This is an error case typically provoked by errors in wiring and should be checked by the application - the state of the NodeID pins has to be valid in a typical execution.
- Both, the current value of the NodeID pins and the values stored in the EEPROM are invalid:
This bootloader will use the default NodeID 0xA, the function returns **IO_E_NODEID_PINS_INVALID**.

If necessary, the application can reproduce this check by using the `modifier` parameter and the `nodeNr` field from the APDB to calculate the actual `nodeid` and comparing it to the value provided by this function.



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page		Related Pages		Data Structures		Files	
File List		Globals					
inc							

IO_PID.h File Reference

[Data Structures](#) | [Macros](#) | [Typedefs](#) | [Functions](#)

Contains the data structure for configuring the PID controller.
[More...](#)

```
#include "IO_Driver.h"
```

Data Structures

```
struct _io_pid_config
```

PID configuration structure. [More...](#)

Macros

```
#define IO_PID_MAX_HANDLES 12U
```

Typedefs

```
typedef struct _io_pid_config IO_PID_CONFIG
```

PID configuration structure.

Functions

IO_ErrorType **IO_PID_SetIntegrator** (**ubyte1** pid_handle,
sbyte4 integrator, **bool** hold)

Sets the integral term of a PID controller.

Detailed Description

Contains the data structure for configuring the PID controller.

Macro Definition Documentation

#define IO_PID_MAX_HANDLES 12U

Number maximal allowed PID handles.

In total 12 PID controllers are supported: 6 for Voltage Outputs and 6 for current-controlled PWM outputs.

Typedef Documentation

typedef struct _io_pid_config IO_PID_CONFIG

PID configuration structure.

Data structure that contains all configuration parameters for PID control.

The following formula is used for the PID:

```
output = Kff/10^3 * setpoint + Kp/10^3 *  
        error + Kd/10^4 * delta(error) + Ki/10^4  
        * sum(error)
```

Please note that Kff and Kp are scaled by a factor of 1,000 and Ki and Kd by a factor of 10,000 to improve the resolution on small gains.

Remarks

- Typical values for Kff are between 10.000 and 15.000
- Typical values for Ki are around 20.000
- Typical values for Kp and Kd are smaller than 1.000

Attention

Please keep in mind that the unit of `output` has to be

- Volts for voltage outputs
- Duty cycle in digits [0 .. 65535] for PWM outputs

Function Documentation

```
IO_ErrorType IO_PID_SetIntegrator ( ubyte1 pid_handle,  
                                     sbyte4 integrator,  
                                     bool   hold  
                                     )
```

Sets the integral term of a PID controller.

Sets the integral term of a PID controller to a given value or freezes it.

Parameters

pid_handle handle of PID controller.

integrator new value of integral term; parameter is ignored if **hold** is `TRUE`

hold `TRUE` to freeze the integral term to the current value. Can be unfrozen by calling again with **hold**=`FALSE`

Returns

`IO_ErrorType`

Return values

IO_E_OK everything fine

IO_E_INVALID_PARAMETER an invalid parameter has been passed

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			
			Macros Enumerations
<h2>IO_Pins.h File Reference</h2>			

Global IO Pin defines for IO Driver. [More...](#)

Macros

```
#define IO_ADC_UBAT IO_PIN_L2
```

```
#define IO_ADC_UBAT_CPU IO_PIN_L1
```

```
#define IO_K15 IO_PIN_K4
```

```
#define IO_ADC_SENSOR_SUPPLY IO_PIN_H3
```

```
#define IO_ADC_NODE_ID_0 IO_PIN_K3
```

```
#define IO_ADC_NODE_ID_1 IO_PIN_J3
```

```
#define IO_ADC_20 IO_PIN_G4
```

```
#define IO_DI_02 IO_PIN_G4
```

```
#define IO_ADC_21 IO_PIN_F4
```

```
#define IO_DI_03 IO_PIN_F4
```

```
#define IO_PWD_00 IO_PIN_E3
```

```
#define IO_ADC_30 IO_PIN_E3
```

```
#define IO_DI_04 IO_PIN_E3
```

```
#define IO_PWD_01 IO_PIN_D3
```

```
#define IO_ADC_31 IO_PIN_D3
```

```
#define IO_DI_05 IO_PIN_D3
```

```
#define IO_PWD_02 IO_PIN_C3
```

```
#define IO_ADC_32 IO_PIN_C3
```

```
#define IO_DI_06 IO_PIN_C3
```

```
#define IO_PWD_03 IO_PIN_B3
```

```
#define IO_ADC_33 IO_PIN_B3
```

```
#define IO_DI_07 IO_PIN_B3
```

```
#define IO_ADC_00 IO_PIN_J4
```

```
#define IO_LED_00 IO_PIN_J4
```

```
#define IO_DI_00 IO_PIN_J4
```

```
#define IO_ADC_01 IO_PIN_H4
```

```
#define IO_LED_01 IO_PIN_H4
```

```
#define IO_DI_01 IO_PIN_H4
```

```
#define IO_ADC_10 IO_PIN_E4
```

```
#define IO_LED_02 IO_PIN_E4
```

```
#define IO_DI_10 IO_PIN_E4
```

```
#define IO_ADC_11 IO_PIN_D4
```

```
#define IO_LED_03 IO_PIN_D4
```

```
#define IO_DI_11 IO_PIN_D4
```

```
#define IO_ADC_12 IO_PIN_C4
```

```
#define IO_LED_04 IO_PIN_C4
```

```
#define IO_DI_12 IO_PIN_C4
```

```
#define IO_ADC_13 IO_PIN_B4
```

```
#define IO_LED_05 IO_PIN_B4
```

```
#define IO_DI_13 IO_PIN_B4
```

```
#define IO_ADC_14 IO_PIN_A4
```

```
#define IO_LED_06 IO_PIN_A4
```

```
#define IO_DI_14 IO_PIN_A4
```

```
#define IO_ADC_15 IO_PIN_A3
```

```
#define IO_LED_07 IO_PIN_A3
```

```
#define IO_DI_15 IO_PIN_A3
```

```
#define IO_PWM_00 IO_PIN_H1
```

```
#define IO_DO_00 IO_PIN_H1
```

```
#define IO_DO_20 IO_PIN_H1
```

```
#define IO_ADC_34 IO_PIN_H1
```

```
#define IO_DI_24 IO_PIN_H1
```

```
#define IO_PWD_20 IO_PIN_H1
```

```
#define IO_PWM_01 IO_PIN_G1
```

```
#define IO_DO_01 IO_PIN_G1
```

```
#define IO_DO_21 IO_PIN_G1
```

```
#define IO_ADC_35 IO_PIN_G1
```

```
#define IO_DI_25 IO_PIN_G1
```

```
#define IO_PWD_21 IO_PIN_G1
```

```
#define IO_PWM_02 IO_PIN_F1
```

```
#define IO_DO_02 IO_PIN_F1
```

```
#define IO_DO_22 IO_PIN_F1
```

```
#define IO_ADC_36 IO_PIN_F1
```

```
#define IO_DI_26 IO_PIN_F1
```

```
#define IO_PWD_10 IO_PIN_F1
```

```
#define IO_PWM_03 IO_PIN_E1
```

```
#define IO_DO_03 IO_PIN_E1
```

```
#define IO_DO_23 IO_PIN_E1
```

```
#define IO_ADC_37 IO_PIN_E1
```

```
#define IO_DI_27 IO_PIN_E1
```

```
#define IO_PWD_11 IO_PIN_E1
```

```
#define IO_PWM_04 IO_PIN_D1
```

```
#define IO_DO_04 IO_PIN_D1
```

```
#define IO_DO_24 IO_PIN_D1
```

```
#define IO_ADC_38 IO_PIN_D1
```

```
#define IO_DI_28 IO_PIN_D1
```

```
#define IO_PWD_12 IO_PIN_D1
```

```
#define IO_PWM_05 IO_PIN_C1
```

```
#define IO_DO_05 IO_PIN_C1
```

```
#define IO_DO_25 IO_PIN_C1
```

```
#define IO_ADC_39 IO_PIN_C1
```

```
#define IO_DI_29 IO_PIN_C1
```

```
#define IO_PWD_13 IO_PIN_C1
```

```
#define IO_PWM_10 IO_PIN_K1
```

```
#define IO_DO_06 IO_PIN_K1
```

```
#define IO_ADC_40 IO_PIN_K1
```

```
#define IO_DI_30 IO_PIN_K1
```

```
#define IO_PWD_22 IO_PIN_K1
```

```
#define IO_PWM_11 IO_PIN_J1
```

```
#define IO_DO_07 IO_PIN_J1
```

```
#define IO_ADC_41 IO_PIN_J1
```

```
#define IO_DI_31 IO_PIN_J1
```

```
#define IO_PWD_23 IO_PIN_J1
```

```
#define IO_SAFETY_SWITCH_0 IO_PIN_B1
```

```
#define IO_DO_10 IO_PIN_B1
```

```
#define IO_ADC_28 IO_PIN_B1
```

```
#define IO_DI_22 IO_PIN_B1
```

```
#define IO_SAFETY_SWITCH_1 IO_PIN_A1
```

```
#define IO_DO_11 IO_PIN_A1
```

```
#define IO_ADC_29 IO_PIN_A1
```

```
#define IO_DI_23 IO_PIN_A1
```

```
#define IO_PVG_00 IO_PIN_K2
```

```
#define IO_VOUT_00 IO_PIN_K2
```

```
#define IO_ADC_22 IO_PIN_K2
```

```
#define IO_DI_16 IO_PIN_K2
```

```
#define IO_DO_30 IO_PIN_K2
```

```
#define IO_PVG_01 IO_PIN_J2
```

```
#define IO_VOUT_01 IO_PIN_J2
```

```
#define IO_ADC_23 IO_PIN_J2
```

```
#define IO_DI_17 IO_PIN_J2
```

```
#define IO_DO_31 IO_PIN_J2
```

```
#define IO_PVG_02 IO_PIN_H2
```

```
#define IO_VOUT_02 IO_PIN_H2
```

```
#define IO_ADC_24 IO_PIN_H2
```

```
#define IO_DI_18 IO_PIN_H2
```

```
#define IO_DO_32 IO_PIN_H2
```

```
#define IO_PVG_03 IO_PIN_G2
```

```
#define IO_VOUT_03 IO_PIN_G2
```

```
#define IO_ADC_25 IO_PIN_G2
```

```
#define IO_DI_19 IO_PIN_G2
```

```
#define IO_DO_33 IO_PIN_G2
```

```
#define IO_PVG_04 IO_PIN_F2
```

```
#define IO_VOUT_04 IO_PIN_F2
```

```
#define IO_ADC_26 IO_PIN_F2
```

```
#define IO_DI_20 IO_PIN_F2
```

```
#define IO_DO_34 IO_PIN_F2
```

```
#define IO_PVG_05 IO_PIN_E2
```

```
#define IO_VOUT_05 IO_PIN_E2
```

```
#define IO_ADC_27 IO_PIN_E2
```

```
#define IO_DI_21 IO_PIN_E2
```

```
#define IO_DO_35 IO_PIN_E2
```

```
#define IO_UART IO_INT_PIN_UART_CH0
```

```
#define IO_ADC_BOARD_TEMP IO_INT_PIN_TEMP
```

```
#define IO_INT_POWERSTAGE_ENABLE IO_INT_PIN_POWERSTAGE_ENABLE
```

```
#define IO_INT_PVG_VOUT_0_ENABLE IO_INT_PIN_PVG_VOUT_0_ENABLE
```

```
#define IO_INT_PVG_VOUT_1_ENABLE IO_INT_PIN_PVG_VOUT_1_ENABLE
```

```
#define IO_CAN_CHANNEL_0 IO_INT_PIN_CAN_CH0
```

```
#define IO_CAN_CHANNEL_1 IO_INT_PIN_CAN_CH1
```

```
#define IO_ADC_5V2 IO_INT_PIN_5V2
```

Enumerations

```
enum IO_PIN {  
    IO_PIN_L2 = 0,  
    IO_PIN_L1 = 1,  
    IO_PIN_K4 = 2,  
    IO_PIN_H3 = 3,  
    IO_PIN_K3 = 4,  
    IO_PIN_J3 = 5,  
};
```

IO_PIN_G4 = 6,
IO_PIN_F4 = 7,
IO_PIN_E3 = 8,
IO_PIN_D3 = 9,
IO_PIN_C3 = 10,
IO_PIN_B3 = 11,
IO_PIN_J4 = 12,
IO_PIN_H4 = 13,
IO_PIN_E4 = 14,
IO_PIN_D4 = 15,
IO_PIN_C4 = 16,
IO_PIN_B4 = 17,
IO_PIN_A4 = 18,
IO_PIN_A3 = 19,
IO_PIN_H1 = 20,
IO_PIN_G1 = 21,
IO_PIN_F1 = 22,
IO_PIN_E1 = 23,
IO_PIN_D1 = 24,
IO_PIN_C1 = 25,
IO_PIN_K1 = 26,
IO_PIN_J1 = 27,
IO_PIN_B1 = 28,
IO_PIN_A1 = 29,
IO_PIN_K2 = 30,
IO_PIN_J2 = 31,
IO_PIN_H2 = 32,
IO_PIN_G2 = 33,
IO_PIN_F2 = 34,
IO_PIN_E2 = 35,
IO_INT_PIN_UART_CH0 = 36,
IO_INT_PIN_DRIVER = 37,
IO_INT_PIN_RTC = 38,
IO_INT_PIN_PERIODIC = 39,
IO_INT_PIN_POWER = 40,
IO_INT_PIN_EEPROM = 41,
IO_INT_DEV_CPU = 42,
IO_INT_PIN_EXT_WD = 43,
IO_INT_PIN_TEMP = 44,
IO_INT_PIN_POWERSTAGE_ENABLE = 45,
IO_INT_PIN_PVG_VOUT_0_ENABLE = 46,
IO_INT_PIN_PVG_VOUT_1_ENABLE = 47,
IO_INT_PIN_CAN_CH0 = 48,
IO_INT_PIN_CAN_CH1 = 49,
IO_INT_PIN_5V2 = 50,

```
IO_INT_PIN_SHIFT_LB_HI = 51,  
IO_INT_PIN_SHIFT_LB_LO = 52,  
IO_INT_PIN_SHIFT1_LB_HI = 53,  
IO_INT_PIN_SHIFT1_LB_LO = 54
```

```
}
```

Enumeration of all (internal and external) pins. [More...](#)

Detailed Description

Global IO Pin defines for IO Driver.

This header file defines the IO Pins as well as the aliases for the IO Pins.

Macro Definition Documentation

#define IO_ADC_00 [IO_PIN_J4](#)

Pin J4, for details and features see [IO_PIN_J4](#)

#define IO_ADC_01 [IO_PIN_H4](#)

Pin H4, for details and features see [IO_PIN_H4](#)

#define IO_ADC_10 [IO_PIN_E4](#)

Pin E4, for details and features see [IO_PIN_E4](#)

#define IO_ADC_11 [IO_PIN_D4](#)

Pin D4, for details and features see [IO_PIN_D4](#)

#define IO_ADC_12 [IO_PIN_C4](#)

Pin C4, for details and features see [IO_PIN_C4](#)

#define IO_ADC_13 [IO_PIN_B4](#)

Pin B4, for details and features see [IO_PIN_B4](#)

#define IO_ADC_14 [IO_PIN_A4](#)

Pin A4, for details and features see [IO_PIN_A4](#)

#define IO_ADC_15 [IO_PIN_A3](#)

Pin A3, for details and features see [IO_PIN_A3](#)

```
#define IO_ADC_20 IO\_PIN\_G4
```

Pin G4, for details and features see [IO_PIN_G4](#)

```
#define IO_ADC_21 IO\_PIN\_F4
```

Pin F4, for details and features see [IO_PIN_F4](#)

```
#define IO_ADC_22 IO\_PIN\_K2
```

Pin K2, for details and features see [IO_PIN_K2](#)

```
#define IO_ADC_23 IO\_PIN\_J2
```

Pin J2, for details and features see [IO_PIN_J2](#)

```
#define IO_ADC_24 IO\_PIN\_H2
```

Pin H2, for details and features see [IO_PIN_H2](#)

```
#define IO_ADC_25 IO\_PIN\_G2
```

Pin G2, for details and features see [IO_PIN_G2](#)

```
#define IO_ADC_26 IO\_PIN\_F2
```

Pin F2, for details and features see [IO_PIN_F2](#)

```
#define IO_ADC_27 IO\_PIN\_E2
```

Pin E2, for details and features see [IO_PIN_E2](#)

```
#define IO_ADC_28 IO\_PIN\_B1
```

Pin B1, for details and features see [IO_PIN_B1](#)

#define IO_ADC_29 [IO_PIN_A1](#)

Pin A1, for details and features see [IO_PIN_A1](#)

#define IO_ADC_30 [IO_PIN_E3](#)

Pin E3, for details and features see [IO_PIN_E3](#)

#define IO_ADC_31 [IO_PIN_D3](#)

Pin D3, for details and features see [IO_PIN_D3](#)

#define IO_ADC_32 [IO_PIN_C3](#)

Pin C3, for details and features see [IO_PIN_C3](#)

#define IO_ADC_33 [IO_PIN_B3](#)

Pin B3, for details and features see [IO_PIN_B3](#)

#define IO_ADC_34 [IO_PIN_H1](#)

Pin H1, for details and features see [IO_PIN_H1](#)

#define IO_ADC_35 [IO_PIN_G1](#)

Pin G1, for details and features see [IO_PIN_G1](#)

#define IO_ADC_36 [IO_PIN_F1](#)

Pin F1, for details and features see [IO_PIN_F1](#)

#define IO_ADC_37 [IO_PIN_E1](#)

Pin E1, for details and features see [IO_PIN_E1](#)

#define IO_ADC_38 [IO_PIN_D1](#)

Pin D1, for details and features see [IO_PIN_D1](#)

#define IO_ADC_39 [IO_PIN_C1](#)

Pin C1, for details and features see [IO_PIN_C1](#)

#define IO_ADC_40 [IO_PIN_K1](#)

Pin K1, for details and features see [IO_PIN_K1](#)

#define IO_ADC_41 [IO_PIN_J1](#)

Pin J1, for details and features see [IO_PIN_J1](#)

#define IO_ADC_5V2 [IO_INT_PIN_5V2](#)

internal pin for measuring the 5.2V Voltage

#define IO_ADC_BOARD_TEMP [IO_INT_PIN_TEMP](#)

internal pin for measuring the board temperature

#define IO_ADC_NODE_ID_0 [IO_PIN_K3](#)

Pin K3, for details and features see [IO_PIN_K3](#)

#define IO_ADC_NODE_ID_1 [IO_PIN_J3](#)

Pin J3, for details and features see [IO_PIN_J3](#)

```
#define IO_ADC_SENSOR_SUPPLY IO\_PIN\_H3
```

Pin H3, for details and features see [IO_PIN_H3](#)

```
#define IO_ADC_UBAT IO\_PIN\_L2
```

Pin L2, for details and features see [IO_PIN_L2](#)

```
#define IO_ADC_UBAT_CPU IO\_PIN\_L1
```

Pin L1, for details and features see [IO_PIN_L1](#)

```
#define IO_CAN_CHANNEL_0 IO\_INT\_PIN\_CAN\_CH0
```

Internal Pin for CAN channel 0

```
#define IO_CAN_CHANNEL_1 IO\_INT\_PIN\_CAN\_CH1
```

Internal Pin for CAN channel 1

```
#define IO_DI_00 IO\_PIN\_J4
```

Pin J4, for details and features see [IO_PIN_J4](#)

```
#define IO_DI_01 IO\_PIN\_H4
```

Pin H4, for details and features see [IO_PIN_H4](#)

```
#define IO_DI_02 IO\_PIN\_G4
```

Pin G4, for details and features see [IO_PIN_G4](#)

```
#define IO_DI_03 IO\_PIN\_F4
```

Pin F4, for details and features see [IO_PIN_F4](#)

#define IO_DI_04 [IO_PIN_E3](#)

Pin E3, for details and features see [IO_PIN_E3](#)

#define IO_DI_05 [IO_PIN_D3](#)

Pin D3, for details and features see [IO_PIN_D3](#)

#define IO_DI_06 [IO_PIN_C3](#)

Pin C3, for details and features see [IO_PIN_C3](#)

#define IO_DI_07 [IO_PIN_B3](#)

Pin B3, for details and features see [IO_PIN_B3](#)

#define IO_DI_10 [IO_PIN_E4](#)

Pin E4, for details and features see [IO_PIN_E4](#)

#define IO_DI_11 [IO_PIN_D4](#)

Pin D4, for details and features see [IO_PIN_D4](#)

#define IO_DI_12 [IO_PIN_C4](#)

Pin C4, for details and features see [IO_PIN_C4](#)

#define IO_DI_13 [IO_PIN_B4](#)

Pin B4, for details and features see [IO_PIN_B4](#)

#define IO_DI_14 [IO_PIN_A4](#)

Pin A4, for details and features see [IO_PIN_A4](#)

#define IO_DI_15 [IO_PIN_A3](#)

Pin A3, for details and features see [IO_PIN_A3](#)

#define IO_DI_16 [IO_PIN_K2](#)

Pin K2, for details and features see [IO_PIN_K2](#)

#define IO_DI_17 [IO_PIN_J2](#)

Pin J2, for details and features see [IO_PIN_J2](#)

#define IO_DI_18 [IO_PIN_H2](#)

Pin H2, for details and features see [IO_PIN_H2](#)

#define IO_DI_19 [IO_PIN_G2](#)

Pin G2, for details and features see [IO_PIN_G2](#)

#define IO_DI_20 [IO_PIN_F2](#)

Pin F2, for details and features see [IO_PIN_F2](#)

#define IO_DI_21 [IO_PIN_E2](#)

Pin E2, for details and features see [IO_PIN_E2](#)

#define IO_DI_22 [IO_PIN_B1](#)

Pin B1, for details and features see [IO_PIN_B1](#)

#define IO_DI_23 [IO_PIN_A1](#)

Pin A1, for details and features see [IO_PIN_A1](#)

#define IO_DI_24 [IO_PIN_H1](#)

Pin H1, for details and features see [IO_PIN_H1](#)

#define IO_DI_25 [IO_PIN_G1](#)

Pin G1, for details and features see [IO_PIN_G1](#)

#define IO_DI_26 [IO_PIN_F1](#)

Pin F1, for details and features see [IO_PIN_F1](#)

#define IO_DI_27 [IO_PIN_E1](#)

Pin E1, for details and features see [IO_PIN_E1](#)

#define IO_DI_28 [IO_PIN_D1](#)

Pin D1, for details and features see [IO_PIN_D1](#)

#define IO_DI_29 [IO_PIN_C1](#)

Pin C1, for details and features see [IO_PIN_C1](#)

#define IO_DI_30 [IO_PIN_K1](#)

Pin K1, for details and features see [IO_PIN_K1](#)

#define IO_DI_31 [IO_PIN_J1](#)

Pin J1, for details and features see [IO_PIN_J1](#)

#define IO_DO_00 [IO_PIN_H1](#)

Pin H1, for details and features see [IO_PIN_H1](#)

#define IO_DO_01 [IO_PIN_G1](#)

Pin G1, for details and features see [IO_PIN_G1](#)

#define IO_DO_02 [IO_PIN_F1](#)

Pin F1, for details and features see [IO_PIN_F1](#)

#define IO_DO_03 [IO_PIN_E1](#)

Pin E1, for details and features see [IO_PIN_E1](#)

#define IO_DO_04 [IO_PIN_D1](#)

Pin D1, for details and features see [IO_PIN_D1](#)

#define IO_DO_05 [IO_PIN_C1](#)

Pin C1, for details and features see [IO_PIN_C1](#)

#define IO_DO_06 [IO_PIN_K1](#)

Pin K1, for details and features see [IO_PIN_K1](#)

#define IO_DO_07 [IO_PIN_J1](#)

Pin J1, for details and features see [IO_PIN_J1](#)

#define IO_DO_10 [IO_PIN_B1](#)

Pin B1, for details and features see [IO_PIN_B1](#)

#define IO_DO_11 [IO_PIN_A1](#)

Pin A1, for details and features see [IO_PIN_A1](#)

#define IO_DO_20 [IO_PIN_H1](#)

Pin H1, for details and features see [IO_PIN_H1](#)

#define IO_DO_21 [IO_PIN_G1](#)

Pin G1, for details and features see [IO_PIN_G1](#)

#define IO_DO_22 [IO_PIN_F1](#)

Pin F1, for details and features see [IO_PIN_F1](#)

#define IO_DO_23 [IO_PIN_E1](#)

Pin E1, for details and features see [IO_PIN_E1](#)

#define IO_DO_24 [IO_PIN_D1](#)

Pin D1, for details and features see [IO_PIN_D1](#)

#define IO_DO_25 [IO_PIN_C1](#)

Pin C1, for details and features see [IO_PIN_C1](#)

#define IO_DO_30 [IO_PIN_K2](#)

Pin K2, for details and features see [IO_PIN_K2](#)

#define IO_DO_31 [IO_PIN_J2](#)

Pin J2, for details and features see [IO_PIN_J2](#)

#define IO_DO_32 [IO_PIN_H2](#)

Pin H2, for details and features see [IO_PIN_H2](#)

#define IO_DO_33 [IO_PIN_G2](#)

Pin G2, for details and features see [IO_PIN_G2](#)

#define IO_DO_34 [IO_PIN_F2](#)

Pin F2, for details and features see [IO_PIN_F2](#)

#define IO_DO_35 [IO_PIN_E2](#)

Pin E2, for details and features see [IO_PIN_E2](#)

#define
IO_INT_POWERSTAGE_ENABLE [IO_INT_PIN_POWERSTAGE_ENABLE](#)

Internal Pin for enabling powerstages

#define IO_INT_PVG_VOUT_0_ENABLE [IO_INT_PIN_PVG_VOUT_0_ENABLE](#)

Internal Pin for enabling PVG group 0 outputs

#define IO_INT_PVG_VOUT_1_ENABLE [IO_INT_PIN_PVG_VOUT_1_ENABLE](#)

Internal Pin for enabling PVG group 1 outputs

#define IO_K15 [IO_PIN_K4](#)

Pin K4, for details and features see [IO_PIN_K4](#)

#define IO_LED_00 [IO_PIN_J4](#)

Pin J4, for details and features see [IO_PIN_J4](#)

#define IO_LED_01 [IO_PIN_H4](#)

Pin H4, for details and features see [IO_PIN_H4](#)

#define IO_LED_02 [IO_PIN_E4](#)

Pin E4, for details and features see [IO_PIN_E4](#)

#define IO_LED_03 [IO_PIN_D4](#)

Pin D4, for details and features see [IO_PIN_D4](#)

#define IO_LED_04 [IO_PIN_C4](#)

Pin C4, for details and features see [IO_PIN_C4](#)

#define IO_LED_05 [IO_PIN_B4](#)

Pin B4, for details and features see [IO_PIN_B4](#)

#define IO_LED_06 [IO_PIN_A4](#)

Pin A4, for details and features see [IO_PIN_A4](#)

#define IO_LED_07 [IO_PIN_A3](#)

Pin A3, for details and features see [IO_PIN_A3](#)

#define IO_PVG_00 [IO_PIN_K2](#)

Pin K2, for details and features see [IO_PIN_K2](#)

#define IO_PVG_01 [IO_PIN_J2](#)

Pin J2, for details and features see [IO_PIN_J2](#)

#define IO_PVG_02 [IO_PIN_H2](#)

Pin H2, for details and features see [IO_PIN_H2](#)

#define IO_PVG_03 [IO_PIN_G2](#)

Pin G2, for details and features see [IO_PIN_G2](#)

#define IO_PVG_04 [IO_PIN_F2](#)

Pin F2, for details and features see [IO_PIN_F2](#)

#define IO_PVG_05 [IO_PIN_E2](#)

Pin E2, for details and features see [IO_PIN_E2](#)

#define IO_PWD_00 [IO_PIN_E3](#)

Pin E3, for details and features see [IO_PIN_E3](#)

#define IO_PWD_01 [IO_PIN_D3](#)

Pin D3, for details and features see [IO_PIN_D3](#)

#define IO_PWD_02 [IO_PIN_C3](#)

Pin C3, for details and features see [IO_PIN_C3](#)

#define IO_PWD_03 [IO_PIN_B3](#)

Pin B3, for details and features see [IO_PIN_B3](#)

#define IO_PWD_10 [IO_PIN_F1](#)

Pin F1, for details and features see [IO_PIN_F1](#)

#define IO_PWD_11 [IO_PIN_E1](#)

Pin E1, for details and features see [IO_PIN_E1](#)

#define IO_PWD_12 [IO_PIN_D1](#)

Pin D1, for details and features see [IO_PIN_D1](#)

#define IO_PWD_13 [IO_PIN_C1](#)

Pin C1, for details and features see [IO_PIN_C1](#)

#define IO_PWD_20 [IO_PIN_H1](#)

Pin H1, for details and features see [IO_PIN_H1](#)

#define IO_PWD_21 [IO_PIN_G1](#)

Pin G1, for details and features see [IO_PIN_G1](#)

#define IO_PWD_22 [IO_PIN_K1](#)

Pin K1, for details and features see [IO_PIN_K1](#)

#define IO_PWD_23 [IO_PIN_J1](#)

Pin J1, for details and features see [IO_PIN_J1](#)

#define IO_PWM_00 [IO_PIN_H1](#)

Pin H1, for details and features see [IO_PIN_H1](#)

#define IO_PWM_01 [IO_PIN_G1](#)

Pin G1, for details and features see [IO_PIN_G1](#)

#define IO_PWM_02 [IO_PIN_F1](#)

Pin F1, for details and features see [IO_PIN_F1](#)

#define IO_PWM_03 [IO_PIN_E1](#)

Pin E1, for details and features see [IO_PIN_E1](#)

#define IO_PWM_04 [IO_PIN_D1](#)

Pin D1, for details and features see [IO_PIN_D1](#)

#define IO_PWM_05 [IO_PIN_C1](#)

Pin C1, for details and features see [IO_PIN_C1](#)

#define IO_PWM_10 [IO_PIN_K1](#)

Pin K1, for details and features see [IO_PIN_K1](#)

#define IO_PWM_11 [IO_PIN_J1](#)

Pin J1, for details and features see [IO_PIN_J1](#)

```
#define IO_SAFETY_SWITCH_0  IO_PIN_B1
```

Pin B1, for details and features see [IO_PIN_B1](#)

```
#define IO_SAFETY_SWITCH_1  IO_PIN_A1
```

Pin A1, for details and features see [IO_PIN_A1](#)

```
#define IO_UART  IO_INT_PIN_UART_CH0
```

Internal Pin for UART channel on the Debug Adapter

```
#define IO_VOUT_00  IO_PIN_K2
```

Pin K2, for details and features see [IO_PIN_K2](#)

```
#define IO_VOUT_01  IO_PIN_J2
```

Pin J2, for details and features see [IO_PIN_J2](#)

```
#define IO_VOUT_02  IO_PIN_H2
```

Pin H2, for details and features see [IO_PIN_H2](#)

```
#define IO_VOUT_03  IO_PIN_G2
```

Pin G2, for details and features see [IO_PIN_G2](#)

```
#define IO_VOUT_04  IO_PIN_F2
```

Pin F2, for details and features see [IO_PIN_F2](#)

```
#define IO_VOUT_05  IO_PIN_E2
```

Pin E2, for details and features see [IO_PIN_E2](#)

Enumeration Type Documentation

enum `IO_PIN`

Enumeration of all (internal and external) pins.

Lists all used and available (virtual) I/O pins for operating functionality as well as for diagnostic purposes.

Enumerator:

`IO_PIN_L2`

Pin L2

main function: High-Side battery supply input. See [ADC input](#) (for battery measurement)

alternate functions: none

`IO_PIN_L1`

Pin L1

main function: CPU battery supply input. See [ADC input](#) (for battery measurement)

alternate functions: none

`IO_PIN_K4`

Pin K4

main function: Terminal 15 input. See [Driver for ECU Power functions](#) for details

alternate functions: none

`IO_PIN_H3`

Pin H3

main function: Sensor supply output. See [ADC input](#) (for feedback measurement)

alternate functions: none

`IO_PIN_K3`

Pin K3 (function NOT available for HY-TTC32 and HY-TTC32S)

main function: Node ID 0 input. See

	ADC input (for feedback measurement) <i>alternate functions:</i> none
<i>IO_PIN_J3</i>	Pin J3 (function NOT available for HY-TTC32 and HY-TTC32S) <i>main function:</i> Node ID 1 input. See ADC input (for feedback measurement) <i>alternate functions:</i> none
<i>IO_PIN_G4</i>	Pin G4 <i>main function:</i> 1-Mode ADC Input <i>alternate functions:</i> Digital Input
<i>IO_PIN_F4</i>	Pin F4 <i>main function:</i> 1-Mode ADC Input <i>alternate functions:</i> Digital Input
<i>IO_PIN_E3</i>	Pin E3 <i>main function:</i> Complex Timer Input (with incremental decoder) <i>alternate functions:</i> Analog Input, Digital Input, Complex Timer Input
<i>IO_PIN_D3</i>	Pin D3 <i>main function:</i> Complex Timer Input (with incremental decoder) <i>alternate functions:</i> Analog Input, Digital Input, Complex Timer Input
<i>IO_PIN_C3</i>	Pin C3 <i>main function:</i> Complex Timer Input <i>alternate functions:</i> Analog Input, Digital Input
<i>IO_PIN_B3</i>	

IO_PIN_J4

Pin B3

main function: **Complex Timer Input**

alternate functions: **Analog Input, Digital Input**

IO_PIN_H4

Pin J4

main function: **4-Mode ADC Input**

alternate functions: **Digital Input, LED Output**

IO_PIN_E4

Pin H4

main function: **4-Mode ADC Input**

alternate functions: **Digital Input, LED Output**

IO_PIN_D4

Pin E4

main function: **3-Mode ADC Input**

alternate functions: **Digital Input, LED Output**

IO_PIN_C4

Pin D4

main function: **3-Mode ADC Input**

alternate functions: **Digital Input, LED Output**

IO_PIN_B4

Pin C4

main function: **3-Mode ADC Input**

alternate functions: **Digital Input, LED Output**

IO_PIN_A4

Pin B4

main function: **3-Mode ADC Input**

alternate functions: **Digital Input, LED Output**

Pin A4

main function: **3-Mode ADC Input**

IO_PIN_A3

alternate functions: **Digital Input, LED Output**

Pin A3

main function: **3-Mode ADC Input**

alternate functions: **Digital Input, LED Output**

IO_PIN_H1

Pin H1

main function: **PWM Output with current measurement**

alternate functions: **Timer Input, High-Side Digital Output (with current measurement), High-Side Digital Output, Analog input, Digital input**

IO_PIN_G1

Pin G1

main function: **PWM Output with current measurement**

alternate functions: **Timer Input, High-Side Digital Output (with current measurement), High-Side Digital Output, Analog input, Digital input**

IO_PIN_F1

Pin F1

main function: **PWM Output with current measurement**

alternate functions: **Timer Input, High-Side Digital Output (with current measurement), High-Side Digital Output, Analog input, Digital input**

IO_PIN_E1

Pin E1

main function: **PWM Output with current measurement**

alternate functions: **Timer Input, High-Side Digital Output (with current measurement), High-Side**

IO_PIN_D1

Digital Output, Analog input,
Digital input

Pin D1

main function: **PWM Output with
current measurement**

alternate functions: **Timer Input,
High-Side Digital Output (with
current measurement), High-Side
Digital Output, Analog input,
Digital input**

IO_PIN_C1

Pin C1

main function: **PWM Output with
current measurement**

alternate functions: **Timer Input,
High-Side Digital Output (with
current measurement), High-Side
Digital Output, Analog input,
Digital input**

IO_PIN_K1

Pin K1

main function: **PWM Output**

alternate functions: **Complex
Timer Input, High-Side Digital
Output, Analog input, Digital
input**

IO_PIN_J1

Pin J1

main function: **PWM Output**

alternate functions: **Complex
Timer Input, High-Side Digital
Output, Analog input, Digital
input**

IO_PIN_B1

Pin B1

main function: **Low-Side Digital
Output**

alternate functions: **Analog Input,
Digital Input**

IO_PIN_A1

Pin A1

main function: **Low-Side Digital Output**

alternate functions: **Analog Input, Digital Input**

IO_PIN_K2

Pin K2

main function: **PVG Output**

alternate functions: **Voltage Output, Analog Input, Digital Input, Push-Pull Digital Outputs**

IO_PIN_J2

Pin J2

main function: **PVG Output**

alternate functions: **Voltage Output, Analog Input, Digital Input, Push-Pull Digital Outputs**

IO_PIN_H2

Pin H2

main function: **PVG Output**

alternate functions: **Voltage Output, Analog Input, Digital Input, Push-Pull Digital Outputs**

IO_PIN_G2

Pin G2

main function: **PVG Output**

alternate functions: **Voltage Output, Analog Input, Digital Input, Push-Pull Digital Outputs**

IO_PIN_F2

Pin F2

main function: **PVG Output**

alternate functions: **Voltage Output, Analog Input, Digital Input, Push-Pull Digital Outputs**

IO_PIN_E2

Pin E2

main function: **PVG Output**

alternate functions: **Voltage**

Output, Analog Input, Digital Input, Push-Pull Digital Outputs

IO_INT_PIN_UART_CH0

Internal Pin for UART module, see [Driver for ECU Power functions](#) for details

IO_INT_PIN_DRIVER

Internal Pin for general Driver, see [General Driver items](#) for details

IO_INT_PIN_RTC

Internal Pin for RTC, see [RTC Driver](#) for details

IO_INT_PIN_PERIODIC

Internal Pin for Periodic Timer, see [RTC Driver](#) for details

IO_INT_PIN_POWER

Internal Pin for Power Driver, see [Driver for ECU Power functions](#) for details

IO_INT_PIN_EEPROM

Internal Pin for EEPROM Driver, see [EEPROM Driver](#) for details

IO_INT_DEV_CPU

General device 'pin' for internal CPU diagnosis

IO_INT_PIN_EXT_WD

Internal device pin for diagnosis of external watchdog

IO_INT_PIN_TEMP

Internal Pin for temperature measurement, see [ADC Driver](#) for details

IO_INT_PIN_POWERSTAGE_ENABLE

Internal Pin for Powerstage enable signal, see [Driver for ECU Power functions](#) for details

<i>IO_INT_PIN_PVG_VOUT_0_ENABLE</i>	Internal Pin for PVG group 0 Output enable signal, see Driver for ECU Power functions for details
<i>IO_INT_PIN_PVG_VOUT_1_ENABLE</i>	Internal Pin for PVG group 1 Output enable signal, see Driver for ECU Power functions for details
<i>IO_INT_PIN_CAN_CH0</i>	Internal Pin for CAN channel 0, see CAN Driver for details
<i>IO_INT_PIN_CAN_CH1</i>	Internal Pin for CAN channel 1, see CAN Driver for details
<i>IO_INT_PIN_5V2</i>	Internal Pin for measurement of +5V2 voltage, see ADC Driver for details
<i>IO_INT_PIN_SHIFT_LB_HI</i>	Internal Pin for measurement shift register 0 loopback voltage (high nibble)
<i>IO_INT_PIN_SHIFT_LB_LO</i>	Internal Pin for measurement shift register 0 loopback voltage (low nibble)
<i>IO_INT_PIN_SHIFT1_LB_HI</i>	Internal Pin for measurement shift register 1 loopback voltage (high nibble)
<i>IO_INT_PIN_SHIFT1_LB_LO</i>	Internal Pin for measurement shift register 1 loopback voltage (low nibble)

Main Page		Related Pages	Data Structures	Files
File List	Globals			
inc				

Functions

IO_POWER.h File Reference

IO Driver functions for Power control. [More...](#)

```
#include "IO_Driver.h"
```

Macros

Power values

Selects power configuration.

```
#define IO_POWER_OFF 0
```

```
#define IO_POWER_ON 1
```

Functions

IO_ErrorType **IO_POWER_Init** (void)
Initialization of the power module driver.

IO_ErrorType **IO_POWER_DeInit** (void)

Deinitializes the power module driver.

IO_ErrorType **IO_POWER_Set** (**IO_PIN** pin, **ubyte1** mode)
Sets a certain mode of a POWER feature.

IO_ErrorType **IO_POWER_Get** (**IO_PIN** pin, **ubyte1** *const state)
Returns the current state of a POWER feature.

IO_ErrorType **IO_POWER_SetK15Threshold** (**ubyte2** threshold)
Sets the threshold in mV below which a low-level is detected on the K15 pin.

Detailed Description

IO Driver functions for Power control.

The Power control functions allow enable/disable the power-stages, the PVG/Voltage-outputs as well as switching off the whole unit.

Power code examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

Power initialization example:

The Power driver does not need an explicit initialization. The Power driver is initialized by **IO_Driver_Init()**.

Power task example:

```
// switch on high-side power stages
IO_POWER_Set (IO_INT_POWERSTAGE_ENABLE,
              IO_POWER_ON);

// send ECU to sleep mode (switch off K15)
IO_POWER_Set (IO_K15, IO_POWER_OFF);
```

Macro Definition Documentation

#define IO_POWER_OFF 0

switch off - deactivates the respective power function

#define IO_POWER_ON 1

switch on - activates the respective power function

Function Documentation

IO_ErrorType IO_POWER_DeInit (void)

Deinitializes the power module driver.

Deinitializes the module. Allows re-initialization by

IO POWER Init()

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_NOT_CONFIGURED module has not been initialized

Remarks

This function disables all powerstage-enable signals (PVG/Vout, PWM/DO) and switches off the power outputs permanently. To switch the outputs to a operational state again, the outputs have to be re-initialized (together with the POWER driver)

Re-initializing the POWER driver is only possible on non-safety ECUs or if the IO-Driver was configured not safety-critical.

```
IO_ErrorType IO_POWER_Get ( IO_PIN pin,
                             ubyte1 *const state
                             )
```

Returns the current state of a POWER feature.

Returns the state of power stage enable, PVG/VOut enable or K15

Parameters

pin pin for which the mode shall be set, one of:

- `IO_INT_POWERSTAGE_ENABLE`
- `IO_INT_PVG_VOUT_0_ENABLE`
- `IO_INT_PVG_VOUT_1_ENABLE`
- `IO_SAFETY_SWITCH_0`
- `IO_SAFETY_SWITCH_1`
- `IO_K15`

state pointer to `ubyte1`, returns the state of the selected feature, one of:

- `IO_POWER_ON`
- `IO_POWER_OFF`

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything
fine

`IO_E_INVALID_CHANNEL_ID`

an invalid
channel
has been
chosen

`IO_E_CHANNEL_NOT_CONFIGURED`

module has
not been
initialized

`IO_E_NULL_POINTER`

a NULL
pointer has
been
passed

IO_E_ADC_INVALID

the ADC
value of the
KL15
sampling is
invalid

IO_E_NO_SAFETY_SWITCH_CONFIGURED

the safety
switch is
not used or
was not
configured

Remarks

- `IO_INT_POWERSTAGE_ENABLE` and `IO_INT_PVG_VOUT_ENABLE` are internal pins.
- `IO_INT_POWERSTAGE_ENABLE` controls the internal powerstage enable signal. Without enabling this signal all high-side power outputs remain low (switched off).
- `IO_INT_PVG_VOUT_ENABLE` controls the internal enable signal for the PVG/Voltage-outputs. Without switching this signal to ON, the PVG/Voltage-outputs will remain deactivated.

Note

Both `IO_INT_POWERSTAGE_ENABLE` and `IO_INT_PVG_VOUT_0_ENABLE/IO_INT_PVG_VOUT_1_ENABLE` are "software-switches", i.e. they are pure software functionality without the use of any hardware-parts.

IO_ErrorType `IO_POWER_Init (void)`

Initialization of the power module driver.

Initialization of power module driver.

- Initializes internal data structure
- disables powerstage enable signal
- disables PVG/VOut enable signal

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_BUSY	the module is already initialized
IO_E_TASK_NO_FREE_SLOTS	No more free slots to setup task function

Remarks

Module is initialized only once. To re-initialize the module, the function **IO_POWER_DeInit()** needs to be called.

```
IO_ErrorType IO_POWER_Set ( IO_PIN pin,
                             ubyte1 mode
                             )
```

Sets a certain mode of a POWER feature.

Parameters

pin pin for which the mode shall be set, one of:

- **IO_INT_POWERSTAGE_ENABLE**
- **IO_INT_PVG_VOUT_0_ENABLE**
- **IO_INT_PVG_VOUT_1_ENABLE**
- **IO_SAFETY_SWITCH_0**
- **IO_SAFETY_SWITCH_1**
- **IO_K15** (for power down)

mode sets a certain mode, one of:

- `IO_POWER_ON`
- `IO_POWER_OFF`

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything
fine

`IO_E_BUSY`

driver is in
startup
phase

`IO_E_INVALID_PARAMETER`

an invalid
parameter
has been
passed

`IO_E_INVALID_DIAG_STATE`

the
instruction
is not
permitted in
the current
diagnostic
state.

`IO_E_CHANNEL_NOT_CONFIGURED`

module has
not been
initialized

`IO_E_DRV_SAFETY_CONF_NOT_CONFIG`

driver is
initialized
as non-
safety

`IO_E_DRIVER_NOT_INITIALIZED`

driver is not
initialized

`IO_E_INVALID_CHANNEL_ID`

an invalid
channel

IO_E_GROUP_CONFLICT

has been
chosen
a pin of this
group has
been
initialized in
a different
mode

IO_E_SW_INTERNAL

trying to set
a PVG/Vout
group
which is not
initialized

IO_E_NO_SAFETY_SWITCH_CONFIGURED

the safety
switch is
not used or
was not
configured

Remarks

- `IO_INT_POWERSTAGE_ENABLE` and `IO_INT_PVG_VOUT_x_ENABLE` are internal pins.
- `IO_INT_POWERSTAGE_ENABLE` controls the internal powerstage enable signal. Without enabling this signal all high-side power outputs remain low (switched off).
- `IO_INT_PVG_VOUT_0_ENABLE` and `IO_INT_PVG_VOUT_1_ENABLE` controls the internal enable signal for each group of the PVG/Voltage-outputs. Without switching this signal to ON, the PVG/Voltage-outputs will remain deactivated.
- `IO_SAFETY_SWITCH_0` and `IO_SAFETY_SWITCH_1` allow to en- and disable both safety switches separately. To be able to switch the safety switches the driver must be initialized as safety and running in main state.

If the high side outputs are used as safety critical, turning off the powerstage by calling `IO_POWER_Set (IO_INT_POWERSTAGE_ENABLE, IO_POWER_OFF)` will activate the safe state and return an error code `DIAG_E_PWM_PERIOD_MISMATCH`.

Note

Both `IO_INT_POWERSTAGE_ENABLE` and `IO_INT_PVG_VOUT_0_ENABLE/IO_INT_PVG_VOUT_1_ENABLE` are "software-switches", i.e. they are pure software functionality without the use of any hardware-parts.

IO_ErrorType

IO_POWER_SetK15Threshold

(`ubyte2` **threshold**)

Sets the threshold in mV below which a low-level is detected on the K15 pin.

Parameters

threshold voltage-threshold below which `IO_POWER_OFF` shall be reported for K15. Range: = 0 .. 2500 (0 .. 2500mV)
Default: 2500mV

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything fine

`IO_E_INVALID_PARAMETER`

an invalid parameter has been passed

`IO_E_CHANNEL_NOT_CONFIGURED`

module has not been initialized



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

Functions

IO_PVG.h File Reference

IO Driver functions for PVG channels. [More...](#)

```
#include "IO_Driver.h"
```

Functions

IO_ErrorType **IO_PVG_Init** (**IO_PIN** pvg_channel, **ubyte2** output_value)
Setup one PVG channel.

IO_ErrorType **IO_PVG_DeInit** (**IO_PIN** pvg_channel)
Deinitializes one PVG output.

IO_ErrorType **IO_PVG_SetOutput** (**IO_PIN** pvg_channel, **ubyte2** output_value)
Sets the output value of one PVG channel.

Detailed Description

IO Driver functions for PVG channels.

Contains all service functions for the PVG (Proportional Valve Group) outputs. Up to 6 channels can be configured: `IO_PVG_00 .. IO_PVG_05`

- The pins for PVG- and Voltage-Outputs are organized in two groups of three:
 - Group 1: `IO_PVG_00/IO_VOUT_00`, `IO_PVG_01/IO_VOUT_01` and `IO_PVG_02/IO_VOUT_02` Group 2: `IO_PVG_03/IO_VOUT_03`, `IO_PVG_04/IO_VOUT_04` and `IO_PVG_05/IO_VOUT_05` Whenever a pin within a group is configured either as PVG or voltage output, all other pins within the same group must be either remain unconfigured, or be configured with the same type. For example, if `IO_PIN_J2` (`IO_PVG_01/IO_VOUT_01`) is configured as PVG output, pins `IO_PIN_K2` (`IO_PVG_00/IO_VOUT_00`) and `IO_PIN_H2` (`IO_PVG_02/IO_VOUT_02`) must either remain unconfigured or used as PVG-outputs. Initializing it as any other output type will result in a `IO_E_GROUP_CONFLICT` error.

- The outputs will only be activated after enabling them via

```
IO_POWER_Set (IO_INT_PVG_VOUT_0_ENABLE,  
              IO_POWER_ON) ;  
IO_POWER_Set (IO_INT_PVG_VOUT_1_ENABLE,  
              IO_POWER_ON) ;
```

After activating the outputs, it is not possible to initialize any further PVG channels.

- Recommended initialization/usage order: 1) Initialize ALL needed PVG-channels. 2) Call `IO_POWER_Set(IO_INT_PVG_VOUT_x_ENABLE, IO_POWER_ON);` 3) Set desired output value with `IO_PVG_SetOutput`.
- After enabling the outputs by calling `IO_POWER_Set`, all configured PVG channels will output the desired value, while all other pins within this group will output 0V constantly.
- When configuring a PVG output, the associated voltage feedback channel will also be configured. If the difference between the measured output (U_{feedback}) and the configured output calculated by $U_{\text{diff}} = \text{output_value} / 100 * U_{\text{BAT}} - U_{\text{feedback}}$ is greater than $\text{abs}(+9.5\text{V})$, the output protection will be activated:
 - If $U_{\text{diff}} > 9.5\text{V}$, the output protection will set the output to 1500 (i.e. 15%) for 1 second.
 - If $U_{\text{diff}} < -9.5\text{V}$, the output protection will set the output to 8000 (i.e. 80%) for 1 second.

Function Documentation

IO_ErrorType IO_PVG_DeInit (IO_PIN pvg_channel)

Deinitializes one PVG output.

Parameters

pvg_channel PVG output (`IO_PVG_00` .. `IO_PVG_05`)

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_INVALID_CHANNEL_ID

the given channel
id does not exist

IO_E_CHANNEL_NOT_CONFIGURED the given channel
is not configured

Remarks

- The following channels form groups. A group always has to be configured as a whole.
 - `IO_PVG_00` .. `IO_PVG_02`
 - `IO_PVG_03` .. `IO_PVG_05`

**IO_ErrorType IO_PVG_Init (IO_PIN pvg_channel,
 ubyte2 output_value
)**

Setup one PVG channel.

Parameters

pvg_channel PVG channel (`IO_PVG_00` .. `IO_PVG_05`)

output_value Output value with which the PVG-channel will be initialized in percent * 100 (1500..8500) of the supply voltage. The configured value will be output after enabling the PVG-outputs via `IO_POWER_Set (IO_INT_PVG_VOUT_x_ENABLE)` until the first call of `IO_PVG_SetOutput`.

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything fine

`IO_E_GROUP_CONFLICT`

configuring the output not allowed due to conflicts with other in/outputs in the same group

`IO_E_INVALID_CHANNEL_ID`

the channel id does not exist

`IO_E_INVALID_PARAMETER`

parameter is out of range

`IO_E_CHANNEL_BUSY`

the ADC input channel is currently used by another function

`IO_E_DRIVER_NOT_INITIALIZED` The common driver init function has not been called before

`IO_E_TASK_NO_FREE_SLOTS` No more free slots to setup task function

`IO_E_SW_INTERNAL` Internal software error

IO_E_CH_CAPABILITY

The ADC capability of this channel has not been activated

Remarks

- The following channels form groups. They have to be configured in the same mode within a group.
 - `IO_PVG_00 .. IO_PVG_02`
 - `IO_PVG_03 .. IO_PVG_05`
- Check the alternate functions of the pins used in each group. A pin can only be configured for one function at a time and it has to be the same function within the group - however mixing DO functionality together with PVG functionality in the same group is a valid configuration and vice versa. The alternate functions can be found at [IO_Pins.h](#)

```
IO_ErrorType IO_PVG_SetOutput ( IO_PIN pvg_channel,  
                                ubyte2 output_value  
                                )
```

Sets the output value of one PVG channel.

Parameters

pvg_channel PVG channel (`IO_PVG_00 .. IO_PVG_05`)

output_value Output value in percent * 100 (1500..8500) of the supply voltage

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything fine

`IO_E_INVALID_CHANNEL_ID`

the channel id does not

IO_E_INVALID_PARAMETER	exist parameter is out of range
IO_E_CHANNEL_BUSY	the ADC input channel is currently used by another function
IO_E_PVG_SHORT_BATTERY	short to UBAT has been detected
IO_E_PVG_SHORT_CIRCUIT	short to GND has been detected
IO_E_PVG_OUTPUT_DISABLED	the PVG output is disabled
IO_E_PROT_ACTIVE	Protection is active and output value was automatically set to a value to protect the outputs.
IO_E_PROT_REENABLE	The IO-Driver signals that the protection was disabled and the output functionality was restored.
IO_E_ADC_INVALID	the ADC returned an invalid result
IO_E_CH_CAPABILITY	the chosen channel does not support the requested feature

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			
Data Structures Typedefs Functions			
<h2>IO_PWD.h File Reference</h2>			

IO Driver functions for timer input channels. [More...](#)

```
#include "IO_Driver.h"
```

Data Structures

struct [_io_pwd_pulse_samples](#)
PWD pulse-width data structure. [More...](#)

struct [_io_pwd_inc_safety_conf](#)
Safety configuration for the Incremental or Counter PWD inputs. [More...](#)

struct [_io_pwd_cplx_safety_conf](#)
Safety configuration for the Complex PWD inputs. [More...](#)

Macros

High- / Low time

Specifies whether the high,low or both(period) time shall be captured

```
#define IO\_PWD\_LOW\_TIME 0
```

```
#define IO\_PWD\_HIGH\_TIME 1
```

```
#define IO\_PWD\_PERIOD\_TIME 2
```

Variable edge

Specify the variable edge of the input signal.

If the rising edge is variable, the frequency is measured between the surrounding falling edges.

```
#define IO\_PWD\_RISING\_VAR 2
```

```
#define IO_PWD_FALLING_VAR 3
```

Counting mode

Specify the counting mode of a incremental channel.

```
#define IO_PWD_INC_2_COUNT 0x03
```

```
#define IO_PWD_INC_1_COUNT 0x01
```

PWD resolution configuration

Resolution of the timer of a complex timer input. Note: the timing measurement is based on a 16bit timer, therefore the product (65535 * resolution) must be greater than the period that shall be measured

```
#define IO_PWD_RESOLUTION_0_2 0x01
```

```
#define IO_PWD_RESOLUTION_0_4 0x02
```

```
#define IO_PWD_RESOLUTION_0_8 0x03
```

```
#define IO_PWD_RESOLUTION_1_6 0x04
```

```
#define IO_PWD_RESOLUTION_3_2 0x05
```

Pull up / down configuration

Pull up/down resistor for the PWD inputs

```
#define IO_PWD_PU 0x01
```

```
#define IO_PWD_PD 0x02
```

edge count

Specify on which edge shall be counted

```
#define IO_PWD_RISING_COUNT 1
```

```
#define IO_PWD_FALLING_COUNT 2
```

```
#define IO_PWD_BOTH_COUNT 3
```

count direction

Specify the counting direction

```
#define IO_PWD_UP_COUNT 0
```

```
#define IO_PWD_DOWN_COUNT 1
```

PWD maximum pulse pulse-width samples

Maximum pulse-width samples that can be stored in the datastructure

IO_PWD_PULSE_SAMPLES

```
#define IO_PWD_MAX_PULSE_SAMPLES 9
```

Typedefs

```
typedef struct  
_io_pwd_pulse_samples IO_PWD_PULSE_SAMPLES  
PWD pulse-width data structure.
```

```
typedef struct  
_io_pwd_inc_safety_conf IO_PWD_INC_SAFETY_CONF  
Safety configuration for the Incremental or Counter PWD  
inputs.
```

```
typedef struct  
_io_pwd_cplx_safety_conf IO_PWD_CPLX_SAFETY_CONF  
Safety configuration for the Complex PWD inputs.
```

Functions

```
IO_ErrorType IO_PWD_IncInit (IO_PIN inc_channel, ubyte1 mode, ubyte2 count_init,  
ubyte1 pupd, const IO_PWD_INC_SAFETY_CONF *const safety_conf)  
Setup single incremental interface.
```

```
IO_ErrorType IO_PWD_IncGet (IO_PIN inc_channel, ubyte2 *const count, ubyte2  
*const adc_value_0, ubyte2 *const adc_value_1, bool *const fresh_0,  
bool *const fresh_1)  
Get the counter value of a incremental interface.
```

```
IO_ErrorType IO_PWD_IncSet (IO_PIN inc_channel, ubyte2 count)  
Set the counter value of a incremental interface.
```

```
IO_ErrorType IO_PWD_IncDeInit (IO_PIN inc_channel)
```

Deinitializes a single incremental interface.

IO_ErrorType **IO_PWD_ComplexInit** (**IO_PIN** timer_channel, **ubyte1** pulse_mode, **ubyte1** freq_mode, **ubyte1** timer_res, **ubyte1** capture_count, **ubyte1** pupd, const **IO_PWD_CPLX_SAFETY_CONF** *const safety_conf)
Setup single timer channel that measures frequency and pulse-width at the same time.

IO_ErrorType **IO_PWD_ComplexGet** (**IO_PIN** timer_channel, **ubyte2** *const frequency, **ubyte4** *const pulse_width, **IO_PWD_PULSE_SAMPLES** *const pulse_samples, **ubyte2** *const duty_cycle, **ubyte2** *const adc_value, **bool** *const fresh)
Get the frequency and the pulse-width from the specified timer channel.

IO_ErrorType **IO_PWD_ComplexDelnit** (**IO_PIN** timer_channel)
Deinitializes a complex PWD input.

IO_ErrorType **IO_PWD_FreqInit** (**IO_PIN** timer_channel, **ubyte1** freq_mode)
Setup single timer channel that measures frequency only.

IO_ErrorType **IO_PWD_PulseInit** (**IO_PIN** timer_channel, **ubyte1** pulse_mode)
Setup single timer channel that measures pulse-width only.

IO_ErrorType **IO_PWD_PulseFreqInit** (**IO_PIN** timer_channel, **ubyte1** capture_mode)
Setup single timer channel that measures pulse-width and frequency.

IO_ErrorType **IO_PWD_FreqGet** (**IO_PIN** timer_channel, **ubyte2** *const frequency)
Get the frequency.

IO_ErrorType **IO_PWD_PulseGet** (**IO_PIN** timer_channel, **ubyte4** *const pulse_width)
Get the pulse-width.

IO_ErrorType **IO_PWD_PulseFreqGet** (**IO_PIN** timer_channel, **ubyte2** *const frequency, **ubyte4** *const pulse_width)
Get the pulse-width.

IO_ErrorType **IO_PWD_FreqDelnit** (**IO_PIN** timer_channel)
Deinitializes a PWD input for frequency measurement. Allows the re-initialization of the input by other functions.

IO_ErrorType **IO_PWD_PulseDelnit** (**IO_PIN** timer_channel)
Deinitializes a PWD input for pulse-width measurement. Allows the re-initialization of the input by other functions.

IO_ErrorType **IO_PWD_PulseFreqDelnit** (**IO_PIN** timer_channel)

De-initializes a PWD input for pulse-width and frequency measurement.
Allows the re-initialization of the input by other functions.

IO_ErrorType **IO_PWD_CountInit** (**IO_PIN** count_channel, **ubyte1** mode, **ubyte1** direction, **ubyte2** count_init, **ubyte1** pupd, **IO_PWD_INC_SAFETY_CONF** const *const safety_conf)
Setup single counter channel.

IO_ErrorType **IO_PWD_CountGet** (**IO_PIN** count_channel, **ubyte2** *const count, **ubyte2** *const adc_value, **bool** *const fresh)
Get the counter value of a single counter channel.

IO_ErrorType **IO_PWD_CountSet** (**IO_PIN** count_channel, **ubyte2** count)
Set the counter value of a single counter channel.

IO_ErrorType **IO_PWD_CountDelnit** (**IO_PIN** count_channel)
Deinitializes a single counter channel.

Detailed Description

IO Driver functions for timer input channels.

Contains all service functions for the PWD (Pulse Width Demodulation). There are two different groups of timer inputs:

- `IO_PWD_00 .. IO_PWD_03` and `IO_PWD_22 .. IO_PWD_23`: Complex timer inputs. Can be configured to measure frequency, pulse-width and duty-cycle at the same time. Furthermore pins `IO_PWD_00` in combination with `IO_PWD_01` can read incremental (relative) encoders. In this case two inputs are reserved for one incremental encoder (clock and direction).
- `IO_PWD_10 .. IO_PWD_13`, `IO_PWD_20 .. IO_PWD_21`: Simple timer inputs. Can be configured to measure either frequency or pulse-width. These inputs evaluate only 2 edges (frequency) or 3 edges (pulse-width).

PWD code examples

Please refer to section [Basic structure of an application](#) for understanding where to place the initialization and task function calls.

PWD initialization examples:

```
// setup frequency measurement input
IO_PWD_FreqInit( IO_PWD_20
                , IO_PWD_FALLING_VAR ); // select falling edge as
variable

// setup pulse measurement input
IO_PWD_PulseInit( IO_PWD_21
                 , IO_PWD_HIGH_TIME ); // measure high time of
the signal

// setup complex timer input (frequency and pulse measurement)
IO_PWD_ComplexInit( IO_PWD_03
                   , IO_PWD_HIGH_TIME           // measure high time
                   , IO_PWD_FALLING_VAR         // select falling
edge as variable
                   , IO_PWD_RESOLUTION_0_8      // set timer
resolution to 0.8us
                   , 8                          // set number of
accumulations
                   , IO_PWD_PU                  // configure pull-up
resistor
                   , NULL );                   // Handle for PID
controller not needed => pass NULL

// setup incremental input
IO_PWD_IncInit( IO_PWD_00
               , IO_PWD_INC_2_COUNT
               , 0x7FFF                          // set initial value for
counter
               , IO_PWD_PU                      // configure pull-up
resistor
               , NULL );                       // Handle for PID
controller not needed => pass NULL
```

PWD task examples:

```
ubyte2 freq_20_val, freq_3_val, duty_3_val, adc_3_val, inc_0_val,
      adc_4_val, adc_5_val;
ubyte4 pulse_21_val, pulse_3_val;
bool adc_3_fresh, adc_4_fresh, adc_5_fresh;
```

```
// read frequency value
IO_PWD_FreqGet( IO_PWD_20
               , &freq_20_val );

// read pulse value
IO_PWD_PulseGet( IO_PWD_21
               , &pulse_21_val );

// read complex timer values (frequency and pulse value)
IO_PWD_ComplexGet( IO_PWD_03
                  , &freq_3_val
                  , &pulse_3_val
                  , NULL // pulse samples not needed
                  , &duty_3_val
                  , &adc_3_val
                  , &adc_3_fresh );

// read incremental counter value
IO_PWD_IncGet ( IO_PWD_00
               , &inc_0_val
               , &adc_4_val
               , &adc_5_val
               , &adc_4_fresh
               , &adc_5_fresh);
```

Macro Definition Documentation

#define IO_PWD_BOTH_COUNT 3

count on both edges

#define IO_PWD_DOWN_COUNT 1

count down

#define IO_PWD_FALLING_COUNT 2

count on a falling edge

#define IO_PWD_FALLING_VAR 3

falling edge of the input signal is the variable one

#define IO_PWD_HIGH_TIME 1

capture the high time of the input signal

#define IO_PWD_INC_1_COUNT 0x01

count only on one edge of the two inputs

#define IO_PWD_INC_2_COUNT 0x03

count on any edge of the two inputs

#define IO_PWD_LOW_TIME 0

capture the low time of the input signal

#define IO_PWD_PD 0x02

pull-down resistor

#define IO_PWD_PERIOD_TIME 2

capture the high and low time of the input signal

#define IO_PWD_PU 0x01

pull-up resistor

#define IO_PWD_RESOLUTION_0_2 0x01

resolution is set to 0.2 us

#define IO_PWD_RESOLUTION_0_4 0x02

resolution is set to 0.4 us

#define IO_PWD_RESOLUTION_0_8 0x03

resolution is set to 0.8 us

#define IO_PWD_RESOLUTION_1_6 0x04

resolution is set to 1.6 us

#define IO_PWD_RESOLUTION_3_2 0x05

resolution is set to 3.2 us

#define IO_PWD_RISING_COUNT 1

count on a rising edge

```
#define IO_PWD_RISING_VAR 2
```

rising edge of the input signal is the variable one

```
#define IO_PWD_UP_COUNT 0
```

count up

Typedef Documentation

typedef struct _io_pwd_cplx_safety_conf IO_PWD_CPLX_SAFETY_CONF

Safety configuration for the Complex PWD inputs.

Stores all relevant safety configuration parameters for the Complex PWD inputs.

typedef struct _io_pwd_inc_safety_conf IO_PWD_INC_SAFETY_CONF

Safety configuration for the Incremental or Counter PWD inputs.

Stores all relevant safety configuration parameters for the Incremental PWD inputs.

typedef struct _io_pwd_pulse_samples IO_PWD_PULSE_SAMPLES

PWD pulse-width data structure.

stores each captured pulse-width for one measurement.

Function Documentation

IO_ErrorType IO_PWD_ComplexDelnit (**IO_PIN** timer_channel)

Deinitializes a complex PWD input.

Parameters

timer_channel Timer channel, one of:

- **IO_PWD_00** .. **IO_PWD_03**
- **IO_PWD_22** .. **IO_PWD_23**

Returns

IO_ErrorType:

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CH_CAPABILITY	The given channel is not a PWD input

Remarks

- The following channels form groups. A group always has to be configured as a whole.
 - **IO_PWD_00** .. **IO_PWD_01**
 - **IO_PWD_02** .. **IO_PWD_03**

IO_ErrorType

IO_PWD_ComplexGet	(IO_PIN	timer_channel,
	ubyte2 *const	frequency,
	ubyte4 *const	pulse_width,
	IO_PWD_PULSE_SAMPLES *const	pulse_samples,
	ubyte2 *const	duty_cycle,
	ubyte2 *const	adc_value,
	bool *const	fresh
)	

Get the frequency and the pulse-width from the specified timer channel.

Parameters

timer_channel Timer channel, one of:

- **IO_PWD_00** .. **IO_PWD_03**

	<ul style="list-style-type: none"> • <code>IO_PWD_22 .. IO_PWD_23</code>
frequency	Accumulated frequency in Hz
pulse_width	Accumulated pulse-width in us
pulse_samples	contains each pulse-width measure sample
duty_cycle	contains the measured duty cycle in percent * 10 (e.g. 200 for 20%).
adc_value	ADC value, range: 0..32780 (in mV; 0..33333 for TTC32 variants)
fresh	Status of the ADC value <ul style="list-style-type: none"> • <code>TRUE</code>: ADC value is fresh • <code>FALSE</code>: ADC value is old

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CH_CAPABILITY</code>	The given channel is either not a PWD input
<code>IO_E_PWD_NOT_FINISHED</code>	not enough edges to accumulate a result
<code>IO_E_PWD_CAPTURE_ERROR</code>	the frequency was too high
<code>IO_E_NULL_POINTER</code>	A NULL pointer has been passed
<code>IO_E_ADC_INVALID</code>	A ADC error occurred
<code>IO_E_PWD_TIMER_OVERFLOW</code>	A timer overflow occurred

Remarks

- The timing measurement is based on a 16bit timer + 8bit overflow timer, therefore the product (65535 * 255 * resolution) must be greater than the period that shall be measured. If this period is greater, the function return `IO_E_PWD_TIMER_OVERFLOW`. The maximum frequency that can be measured is around 10kHz (restricted by low pass filtering)
- The parameters `adc_value` and `fresh` are optional. If not needed, these parameters can be set NULL to ignore them. If the parameters should be calculated one has to provide both, a valid pointer to the location of the `adc_value` and the fresh indication.
- if each individual measured pulse-width sample is not needed, the parameter `pulse_samples` should be set to `NULL`
- The parameter `duty_cycle` contains the duty cycle of the active signal part.
 - If `pulse_mode` is `IO_PWD_HIGH_TIME`, `duty_cycle` contains the duty cycle of the high time
 - If `pulse_mode` is `IO_PWD_LOW_TIME`, `duty_cycle` contains the duty cycle of the low time

Attention

- The calculation of the parameter `duty_cycle` is heavily time consuming due to the slow division unit of the XC2000, especially when the pulse-width is greater or equal 0.4294967s. It is recommended to set this parameter to `NULL` if it is not needed (the calculation of the parameter `duty_cycle` will then be skipped)!

Remarks

- The parameter `pulse_width` contains the
 - high time of the measured signal in microseconds, if `pulse_mode` was set to `IO_PWD_HIGH_TIME`
 - low time of the measured signal in microseconds, if `pulse_mode` was set to `IO_PWD_LOW_TIME`
 - period-time of the measured signal in microseconds, if `pulse_mode` was set to `IO_PWD_PERIOD_TIME`

Attention

- The parameter `frequency` only shows integral frequency values. For input frequencies smaller than 1Hz this value is set to 0 and only the parameters pulse-width and `duty_cycle` are valid.

IO_ErrorType

```
IO_PWD_ComplexInit ( IO_PIN          timer_channel,
                    ubyte1           pulse_mode,
                    ubyte1           freq_mode,
                    ubyte1           timer_res,
                    ubyte1           capture_count,
                    ubyte1           pupd,
                    const IO_PWD_CPLX_SAFETY_CONF *const safety_conf
                    )
```

Setup single timer channel that measures frequency and pulse-width at the same time.

Parameters

timer_channel Timer channel, one of:

- `IO_PWD_00 .. IO_PWD_03`
- `IO_PWD_22 .. IO_PWD_23`

pulse_mode Specify which pulse time to measure:

- `IO_PWD_HIGH_TIME`: configuration to measure pulse-high-time
- `IO_PWD_LOW_TIME`: configuration to measure pulse-low-time
- `IO_PWD_PERIOD_TIME`: configuration to measure pulse-high and low-time (Period). The parameter `pulse_width` of the function `IO_PWD_ComplexGet` will contain the period time instead of the pulse-width.

freq_mode	Specify the variable edge <ul style="list-style-type: none"> <code>IO_PWD_RISING_VAR</code>: rising edge is variable this means that frequency is measured on falling edges <code>IO_PWD_FALLING_VAR</code>: falling edge is variable this means that frequency is measured on rising edges
timer_res	Specify the timer resolution (only for <code>IO_PWD_00</code> .. <code>IO_PWD_03</code>) <ul style="list-style-type: none"> <code>IO_PWD_RESOLUTION_0_2</code>: 0.2us <code>IO_PWD_RESOLUTION_0_4</code>: 0.4us <code>IO_PWD_RESOLUTION_0_8</code>: 0.8us <code>IO_PWD_RESOLUTION_1_6</code>: 1.6us <code>IO_PWD_RESOLUTION_3_2</code>: 3.2us
capture_count	Number of frequency/pulse-width measurements that will be accumulated (0..8)
pupd	Specify which pull-resistor to use (only for <code>IO_PWD_00</code> .. <code>IO_PWD_03</code>) <ul style="list-style-type: none"> <code>IO_PWD_PU</code>: pull-up resistor to 5V <code>IO_PWD_PD</code>: pull-down resistor to ground
safety_conf	Safety configuration.

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_BUSY</code>	the channel is currently used by another function
<code>IO_E_INVALID_CHANNEL_ID</code>	the channel id does not exist
<code>IO_E_INVALID_PARAMETER</code>	parameter capture_count, timebase or mode is out of range
<code>IO_E_CH_CAPABILITY</code>	The PWD-Complex capability of this channel has not been activated
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	The common driver init function has not been called before
<code>IO_E_GROUP_CONFLICT</code>	configuring the output not allowed due to conflicts with other in/outputs in the same group
<code>IO_E_SAFETY_NOT_SUPPORTED</code>	the given channel does not support to be configured safety critical
<code>IO_E_SW_INTERNAL</code>	an internal software error occurred
<code>IO_E_INVALID_SAFETY_CONFIG</code>	one or more parameters of the safety configuration are invalid (out of range)
<code>IO_E_DRV_SAFETY_CONF_NOT_CONFIG</code>	the driver has not been initialized as safety device - therefore the safety feature is not available for this channel

Remarks

A channel that is initialized with this function can retrieve frequency and duty cycle by calling the function: `IO_PWD_ComplexGet()`

The timing measurement is based on a 16bit timer + 8bit overflow timer, therefore the product ($65535 * 255 * \text{resolution}$) must be greater than the period that shall be measured. If this period is greater, the function will return `IO_E_PWD_TIMER_OVERFLOW`. The maximum frequency that can be measured is around 10kHz (restricted by low pass filtering)

The parameters `timer_res` and `pupd` are only consired for `IO_PWD_00 .. IO_PWD_03`. When configuring `IO_PWD_22 .. IO_PWD_23` these parameters are ignored.

If `capture_count = 0`, the driver accumulates all the measurements captured in the last round. Note: In this mode at least 4 edges are required for one measurement. If `capture_count = 1..8`, the driver captures exactly as many measurements are given in the parameter `capture_count`. Note: Until not all configured samples are captured, the driver don't return a value.

The channels `IO_PWD_22` and `IO_PWD_23` are not allowed to be configured safety critical.

The resolution of the channels `IO_PWD_22` and `IO_PWD_23` is fixed to 0.2 microseconds (=200ns).

- The following channels form groups. They have to be configured in the same mode within a group.
 - `IO_PWD_00 .. IO_PWD_01`
 - `IO_PWD_02 .. IO_PWD_03`
- Check the alternate functions of the pins used in each group. A pin can only be configured for one function at a time and it has to be the same function within the group. The alternate functions can be found at [IO_Pins.h](#)

`IO_ErrorType IO_PWD_CountDeInit (IO_PIN count_channel)`

Deinitializes a single counter channel.

Parameters

count_channel Counter channel (`IO_PWD_01`)

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	The given channel is not a incremental timer input

Remarks

Only channel `IO_PWD_01` can be used for counting functionality

```
IO_ErrorType IO_PWD_CountGet ( IO_PIN      count_channel,  
                               ubyte2 *const count,  
                               ubyte2 *const adc_value,  
                               bool *const  fresh  
                               )
```

Get the counter value of a single counter channel.

Parameters

- count_channel** Counter channel (`IO_PWD_01`)
- count** Value of counter (0..65535)
- adc_value** ADC value, Range: 0..32780 (in mV; 0..33333 for TTC32 variants)
- fresh** Status of the ADC value
- `TRUE`: ADC value is fresh
 - `FALSE`: ADC value is old

Returns

`IO_ErrorType`:

Return values

- `IO_E_OK` everything fine
- `IO_E_CHANNEL_NOT_CONFIGURED` the given channel is not configured
- `IO_E_INVALID_CHANNEL_ID` the given channel id does not exist
- `IO_E_CH_CAPABILITY` The given channel is not a incremental timer input
- `IO_E_NULL_POINTER` a NULL pointer has been passed to the function
- `IO_E_ADC_INVALID` the ADC value is invalid

Remarks

Only channel `IO_PWD_01` can be used for counting functionality

The parameters `adc_value` and `fresh` are optional. If not needed, these parameters can be set `NULL` to ignore them. If the parameters should be calculated one has to provide both, a valid pointer to the location of the `adc_value` and the `fresh` indication.

```
IO_ErrorType  
IO_PWD_CountInit ( IO_PIN      count_channel,  
                   ubyte1      mode,  
                   ubyte1      direction,
```

```

        ubyte2
        ubyte1
        IO_PWD_INC_SAFETY_CONF const *const count_init,
        pupd,
        safety_conf
    )

```

Setup single counter channel.

Parameters

- count_channel** Counter channel (`IO_PWD_01`)
- mode** Specify on wich edge shall be count
- `IO_PWD_RISING_COUNT`: count on a rising edge
 - `IO_PWD_FALLING_COUNT`: count on a falling edge
 - `IO_PWD_BOTH_COUNT`: count on a both edges
- direction** Specify the counting direction
- `IO_PWD_UP_COUNT`: counts up
 - `IO_PWD_DOWN_COUNT`: counts down
- count_init** Init value of counter (0..65535)
- pupd** Specify which pull-resistor to use
- `IO_PWD_PU`: pull-up resistor to 5V
 - `IO_PWD_PD`: pull-down resistor to ground
- safety_conf** Safety configuration

Returns

`IO_ErrorType`:

Return values

- | | |
|--|---|
| <code>IO_E_OK</code> | everything fine |
| <code>IO_E_CHANNEL_BUSY</code> | the channel is currently used by another function |
| <code>IO_E_INVALID_CHANNEL_ID</code> | the channel id does not exist |
| <code>IO_E_INVALID_PARAMETER</code> | a parameter is out of range |
| <code>IO_E_CH_CAPABILITY</code> | The PWD-Inc capability of this channel has not been activated |
| <code>IO_E_DRIVER_NOT_INITIALIZED</code> | The common driver init function has not been called before |
| <code>IO_E_SW_INTERNAL</code> | Internal SW malfunction |
| <code>IO_E_INVALID_SAFETY_CONFIG</code> | one or more parameters of the safety configuration are invalid (out of range) |
| <code>IO_E_SAFETY_NOT_SUPPORTED</code> | the given channel does not support safety features |
| <code>IO_E_DRV_SAFETY_CONF_NOT_CONFIG</code> | the driver has not been initialized as safety device - therefore the safety |

feature is not available for this channel

Remarks

- Only channel `IO_PWD_01` can be used for counting functionality. `IO_PWD_00` must not be configured.
- Check the alternate functions of the pin. A pin can only be configured for one function at a time. The alternate functions can be found at [IO_Pins.h](#)

```
IO_ErrorType IO_PWD_CountSet ( IO_PIN count_channel,  
                                ubyte2 count  
                                )
```

Set the counter value of a single counter channel.

Parameters

count_channel Counter channel (`IO_PWD_01`)
count New value to set of incremental counter (0..65535)

Returns

IO_ErrorType:

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CH_CAPABILITY	The given channel is not a incremental timer input

Remarks

Only channel `IO_PWD_01` can be used for counting functionality

```
IO_ErrorType IO_PWD_FreqDeInit ( IO_PIN timer_channel )
```

Deinitializes a PWD input for frequency measurement. Allows the re-initialization of the input by other functions.

Parameters

timer_channel Timer channel, one of:

- `IO_PWD_10` .. `IO_PWD_13`
- `IO_PWD_20` .. `IO_PWD_21`

Returns

IO_ErrorType:

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CH_CAPABILITY	The given channel is not a PWD input

```
IO_ErrorType IO_PWD_FreqGet ( IO_PIN timer_channel,  
                             ubyte2 *const frequency  
                             )
```

Get the frequency.

Parameters

timer_channel Timer channel, one of:

- **IO_PWD_10** .. **IO_PWD_13**
- **IO_PWD_20** .. **IO_PWD_21**

frequency Measured frequency in Hz

Returns

IO_ErrorType:

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CH_CAPABILITY	The given channel is not a PWD input
IO_E_PWD_CAPTURE_ERROR	frequency too high
IO_E_NULL_POINTER	A NULL pointer has been passed
IO_E_PWD_HIGH_LEVEL	only a constant high level is detected
IO_E_PWD_LOW_LEVEL	only a constant low level is detected
IO_E_PWD_NOT_FINISHED	not enough edges to accumulate a result

Remarks

The lowest frequency that can be measured is 10Hz plus a jitter in the size of the cycle time. If the signal has a lower frequency, the function returns a frequency value of 0 and a return a status of **IO_E_PWD_HIGH_LEVEL** or **IO_E_PWD_LOW_LEVEL** after 104ms. The maximum frequency that can be measured is around 10KHz (restricted by low pass filtering)

```
IO_ErrorType IO_PWD_FreqInit ( IO_PIN timer_channel,  
                              ubyte1 freq_mode  
                              )
```

Setup single timer channel that measures frequency only.

Parameters

timer_channel Timer channel, one of:

- `IO_PWD_10 .. IO_PWD_13`
- `IO_PWD_20 .. IO_PWD_21`

freq_mode Specify the variable edge

- `IO_PWD_RISING_VAR`: rising edge is variable
- `IO_PWD_FALLING_VAR`: falling edge is variable

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_BUSY</code>	the input channel is currently used by another function
<code>IO_E_INVALID_CHANNEL_ID</code>	the input channel id does not exist
<code>IO_E_INVALID_PARAMETER</code>	parameter mode is out of range
<code>IO_E_CH_CAPABILITY</code>	The PWD-Freq capability of this channel has not been activated
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	The common driver init function has not been called before

Remarks

The resolution of the channels `IO_PWD_10 .. IO_PWD_13`, `IO_PWD_20` and `IO_PWD_21` is fixed to 1.6 microseconds.

`IO_ErrorType IO_PWD_IncDelnit (IO_PIN inc_channel)`

Deinitializes a single incremental interface.

Parameters

inc_channel Incremental channel (`IO_PWD_00 .. IO_PWD_01`)

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	The given channel is not a incremental timer input

Remarks

Use the primary channel (same channel that is used in the init function).

```
IO_ErrorType IO_PWD_IncGet ( IO_PIN      inc_channel,  
                             ubyte2 *const count,  
                             ubyte2 *const adc_value_0,  
                             ubyte2 *const adc_value_1,  
                             bool *const  fresh_0,  
                             bool *const  fresh_1  
                             )
```

Get the counter value of a incremental interface.

Parameters

- inc_channel** Incremental channel (`IO_PWD_00 .. IO_PWD_01`)
- count** Value of incremental counter (0..65535)
- adc_value_0** ADC value channel 0, Range: 0..32780 (in mV; 0..33333 for TTC32 variants)
- adc_value_1** ADC value channel 1, Range: 0..32780 (in mV; 0..33333 for TTC32 variants)
- fresh_0** Status of the ADC value channel 0
- `TRUE`: ADC value is fresh
 - `FALSE`: ADC value is old
- fresh_1** Status of the ADC value channel 1
- `TRUE`: ADC value is fresh
 - `FALSE`: ADC value is old

Returns

`IO_ErrorType`:

Return values

- `IO_E_OK` everything fine
- `IO_E_CHANNEL_NOT_CONFIGURED` the given channel is not configured
- `IO_E_INVALID_CHANNEL_ID` the given channel id does not exist
- `IO_E_CH_CAPABILITY` The given channel is not a incremental timer input
- `IO_E_NULL_POINTER` a NULL pointer has been passed to the function
- `IO_E_ADC_INVALID` the ADC value is invalid

Remarks

Use the primary channel (same channel that is used in the init function).

The parameters `adc_value_0`, `adc_value_1`, `fresh_0` and `fresh_1` are optional. If not needed, these parameters can be set `NULL` to ignore them. If the parameters should be calculated one has to provide both, a valid pointer to the location of the adc value and the fresh indication for both input channels!

IO_ErrorType

```
IO_PWD_IncInit ( IO_PIN
                  ubyte1
                  ubyte2
                  ubyte1
                  const IO_PWD_INC_SAFETY_CONF *const
                )
                inc_channel,
                mode,
                count_init,
                pupd,
                safety_conf
```

Setup single incremental interface.

Parameters

- inc_channel** Incremental channel (`IO_PWD_00` or `IO_PWD_01`)
- mode** Defines the counter behavior
- `IO_PWD_INC_2_COUNT`: Counts up/down on any edge of the two inputs
 - `IO_PWD_INC_1_COUNT`: Counts up/down on any edge of one channel (`IO_PWD_00` or `IO_PWD_01`)
- count_init** Init value of incremental counter (0..65535)
- pupd** Specify which pull-resistor to use
- `IO_PWD_PU`: pull-up resistor to 5V
 - `IO_PWD_PD`: pull-down resistor to ground
- safety_conf** Safety configuration

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_BUSY</code>	the channel is currently used by another function
<code>IO_E_INVALID_CHANNEL_ID</code>	the channel id does not exist
<code>IO_E_INVALID_PARAMETER</code>	a parameter is out of range
<code>IO_E_CH_CAPABILITY</code>	The PWD-Inc capability of this channel has not been activated
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	The common driver init function has not been called before
<code>IO_E_SW_INTERNAL</code>	Internal SW malfunction
<code>IO_E_INVALID_SAFETY_CONFIG</code>	one or more parameters of the

IO_E_SAFETY_NOT_SUPPORTED

safety configuration are invalid (out of range)

the given channel does not support safety features

IO_E_DRV_SAFETY_CONF_NOT_CONFIG

the driver has not been initialized as safety device - therefore the safety feature is not available for this channel

Remarks

[IO_PWD_00](#) in combination with [IO_PWD_01](#) defines the incremental interface. Both PWD channels are used (needed) for the incremental interface.

The given channel is the primary channel. All further operations (step and deinit functions) must be performed with this channel.

- Check the alternate functions of the pins used. A pin can only be configured for one function at a time. The alternate functions can be found at [IO_Pins.h](#)

```
IO_ErrorType IO_PWD_IncSet ( IO_PIN inc_channel,  
                             ubyte2 count  
                             )
```

Set the counter value of a incremental interface.

Parameters

inc_channel Incremental channel ([IO_PWD_00](#) .. [IO_PWD_01](#))

count New value to set of incremental counter (0..65535)

Returns

[IO_ErrorType](#):

Return values

[IO_E_OK](#)

everything fine

[IO_E_CHANNEL_NOT_CONFIGURED](#)

the given channel is not configured

[IO_E_INVALID_CHANNEL_ID](#)

the given channel id does not exist

[IO_E_CH_CAPABILITY](#)

The given channel is not a incremental timer input

Remarks

Use the primary channel (same channel that is used in the init function).

```
IO_ErrorType IO_PWD_PulseDeInit ( IO_PIN timer_channel )
```

Deinitializes a PWD input for pulse-width measurement. Allows the re-initialization of the input by other functions.

Parameters

timer_channel Timer channel, one of:

- `IO_PWD_10 .. IO_PWD_13`
- `IO_PWD_20 .. IO_PWD_21`

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	The given channel is not a PWD input

`IO_ErrorType IO_PWD_PulseFreqDeInit (IO_PIN timer_channel)`

De-initializes a PWD input for pulse-width and frequency measurement. Allows the re-initialization of the input by other functions.

Parameters

timer_channel Timer channel , one of:

- `IO_PWD_10 .. IO_PWD_13`
- `IO_PWD_20 .. IO_PWD_21`

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	The given channel is not a PWD input

`IO_ErrorType IO_PWD_PulseFreqGet (IO_PIN timer_channel,
ubyte2 *const frequency,
ubyte4 *const pulse_width
)`

Get the pulse-width.

Parameters

timer_channel Timer channel, one of:

- `IO_PWD_10 .. IO_PWD_13`
- `IO_PWD_20 .. IO_PWD_21`

frequency Measured frequency in Hz (optional parameter)

pulse_width Measured pulse-width in us (optional parameter)

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CH_CAPABILITY</code>	The given channel is not a PWD input
<code>IO_E_PWD_CAPTURE_ERROR</code>	frequency too high
<code>IO_E_PWD_HIGH_LEVEL</code>	only a constant high level is detected
<code>IO_E_PWD_LOW_LEVEL</code>	only a constant low level is detected
<code>IO_E_PWD_NOT_FINISHED</code>	not enough edges to accumulate a result

Remarks

- The largest pulse that can be measured is 100ms plus a jitter in the size of the cycle time. If the signal has larger pulses, the function returns a pulse measurement value of 0 and a return a status of `IO_E_PWD_HIGH_LEVEL` or `IO_E_PWD_LOW_LEVEL`. The maximum frequency that can be measured is around 10KHz (restricted by low pass filtering)

Attention

- This function has to be called cyclically with a cycle time smaller than 100ms. Cycle times greater 100ms are not allowed for this function!

```
IO_ErrorType IO_PWD_PulseFreqInit ( IO_PIN timer_channel,  
                                     uint8 capture_mode  
                                     )
```

Setup single timer channel that measures pulse-width and frequency.

Parameters

timer_channel Timer channel, one of:

- `IO_PWD_10 .. IO_PWD_13`
- `IO_PWD_20 .. IO_PWD_21`

capture_mode Capture mode

- `IO_PWD_HIGH_TIME`:
 - measure pulse-high-time and

- measure frequency from falling to falling edge
- `IO_PWD_LOW_TIME`: configuration to measure
 - measure pulse-low-time and
 - measure frequency from rising to rising edge

Returns

`IO_ErrorType`:

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_BUSY</code>	the input channel is currently used by another function
<code>IO_E_INVALID_CHANNEL_ID</code>	the input channel id does not exist
<code>IO_E_INVALID_PARAMETER</code>	parameter mode is out of range
<code>IO_E_CH_CAPABILITY</code>	The PWD-Pulse capability of this channel has not been activated
<code>IO_E_DRIVER_NOT_INITIALIZED</code>	The common driver init function has not been called before

Remarks

The corresponding step function `IO_PWD_PulseFreqGet` has to be called cyclically with a cycle time smaller than 100ms

The resolution of the channels `IO_PWD_10 .. IO_PWD_13`, `IO_PWD_20` and `IO_PWD_21` is fixed to 1.6 microseconds.

```
IO_ErrorType IO_PWD_PulseGet ( IO_PIN          timer_channel,
                               ubyte4 *const pulse_width
                               )
```

Get the pulse-width.

Parameters

timer_channel Timer channel, one of:

- `IO_PWD_10 .. IO_PWD_13`
- `IO_PWD_20 .. IO_PWD_21`

pulse_width Measured pulse-width in us

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist

IO_E_CH_CAPABILITY	The given channel is not a PWD input
IO_E_PWD_CAPTURE_ERROR	frequency too high
IO_E_NULL_POINTER	A NULL pointer has been passed
IO_E_PWD_HIGH_LEVEL	only a constant high level is detected
IO_E_PWD_LOW_LEVEL	only a constant low level is detected
IO_E_PWD_NOT_FINISHED	not enough edges to accumulate a result

Remarks

The largest pulse that can be measured is 100ms plus a jitter in the size of the cycle time. If the signal has larger pulses, the function returns a pulse measurement value of 0 and a return a status of **IO_E_PWD_HIGH_LEVEL** or **IO_E_PWD_LOW_LEVEL** after 104ms. The maximum frequency that can be measured is around 10KHz (restricted by low pass filtering)

```
IO_ErrorType IO_PWD_PulseInit ( IO_PIN timer_channel,
                                ubyte1 pulse_mode
                                )
```

Setup single timer channel that measures pulse-width only.

Parameters

timer_channel Timer channel, one of:

- **IO_PWD_10** .. **IO_PWD_13**
- **IO_PWD_20** .. **IO_PWD_21**

pulse_mode Capture mode

- **IO_PWD_HIGH_TIME**: configuration to measure pulse-high-time
- **IO_PWD_LOW_TIME**: configuration to measure pulse-low-time

Returns

IO_ErrorType:

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_BUSY	the input channel is currently used by another function
IO_E_INVALID_CHANNEL_ID	the input channel id does not exist
IO_E_INVALID_PARAMETER	parameter mode is out of range
IO_E_CH_CAPABILITY	The PWD-Pulse capability of this channel has not been activated
IO_E_DRIVER_NOT_INITIALIZED	The common driver init function has not been called before

Remarks

The resolution of the channels `IO_PWD_10 .. IO_PWD_13`, `IO_PWD_20` and `IO_PWD_21` is fixed to 1.6 microseconds.

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

[Data Structures](#) | [Typedefs](#) | [Functions](#)

IO_PWM.h File Reference

IO Driver functions for PWM channels. [More...](#)

```
#include "IO_Driver.h" #include "IO_RTC.h"
#include "IO_PID.h"
```

Data Structures

struct [_io_pwm_current_queue](#)
PWM current measurement queue. [More...](#)

struct [_io_pwm_safety_conf](#)
Safety configuration for the PWM outputs. [More...](#)

struct [_io_pwm_current_safety_conf](#)
Safety configuration for the PWM outputs. [More...](#)

Macros

Size of current queue

Size of the queue holding the measured values of the equidistant current measurement

```
#define IO\_PWM\_CURRENT\_QUEUE\_MAX 6U
```

Typedefs

typedef struct
[_io_pwm_current_queue](#) [IO_PWM_CURRENT_QUEUE](#)
PWM current measurement queue.

typedef struct [_io_pwm_safety_conf](#) [IO_PWM_SAFETY_CONF](#)
Safety configuration for the PWM outputs.

typedef struct
[_io_pwm_current_safety_conf](#) [IO_PWM_CURRENT_SAFETY_CONF](#)
Safety configuration for the PWM outputs.

Functions

IO_ErrorType **IO_PWM_Init** (**IO_PIN** pwm_channel, **ubyte2** frequency, **bool** polarity, **bool** diag_margin, **ubyte2** overload_limit, const **IO_PWM_SAFETY_CONF** *const safety_conf)
Setup single PWM output.

IO_ErrorType **IO_PWM_CurrentInit** (**IO_PIN** pwm_channel, **ubyte2** frequency, const **IO_PID_CONFIG** *const pid_config, **ubyte1** pwm_period_multi, **ubyte1** *const pid_handle, const **IO_PWM_CURRENT_SAFETY_CONF** *const safety_conf)
Setup a current controlled PWM output.

IO_ErrorType **IO_PWM_CurrentDeInit** (**IO_PIN** pwm_channel)
Deinitializes a current-controlled PWM output. Allows the re-initialization of the output by other functions.

IO_ErrorType **IO_PWM_DeInit** (**IO_PIN** pwm_channel)
Deinitializes a PWM output. Allows the re-initialization of the output by other functions.

IO_ErrorType **IO_PWM_SetDuty** (**IO_PIN** pwm_channel, **ubyte2** duty_cycle, **ubyte4** *const duty_cycle_fb)
Set the duty cycle for a PWM channel.

IO_ErrorType **IO_PWM_SetCur** (**IO_PIN** pwm_channel, **ubyte2** current, **ubyte4** *const duty_cycle_fb)
Sets the current for a current-controlled PWM channel.

IO_ErrorType **IO_PWM_GetCur** (**IO_PIN** pwm_channel, **ubyte2** *const current, **bool** *const fresh)
Returns the measured current of the given channel.

IO_ErrorType **IO_PWM_GetCurQueue** (**IO_PIN** pwm_channel, **IO_PWM_CURRENT_QUEUE** *const current_queue)
Returns the measured current values since the last call of the given channel.

Detailed Description

IO Driver functions for PWM channels.

Contains all service functions for the PWM (Pulse Width Modulation). Up to 8 channels can be configured:

- `IO_PWM_00 .. IO_PWM_05`: High-side PWM outputs with current-measurement
- `IO_PWM_10 .. IO_PWM_11`: High-side PWM outputs with overcurrent-monitoring.
- Not all PWM channels can have their own frequency time base.
 - `IO_PWM_00`, `IO_PWM_01`, `IO_PWM_10` and `IO_PWM_11` have their own frequency time base (variable frequency)
 - `IO_PWM_02`, `IO_PWM_03`, `IO_PWM_04` and `IO_PWM_05` share one frequency time base. It's only allowed to configure these outputs with the same PWM frequency.
- The associated precise current-measurement will be configured for the channels `IO_PWM_00 .. IO_PWM_05`. These PWM outputs will be switched off if the the continuous current is above 3.0A after a specific amount of time (1s) and switched off immediately if the current exceeds 4.0A. The step function for retrieving the current is `IO_PWM_GetCur()`.
- The associated overcurrent-monitoring will be configured for the channels `IO_PWM_10 .. IO_PWM_11`. These PWM outputs will be switched off if the the continuous current is above 3.0A after a specific amount of time (1s). The outputs allow overcurrent for 1 second to allow switching of light-bulbs with high inrush current.
- Additionally, current-controlled PWM outputs can be configured with `IO_PWM_CurrentInit`. These outputs are controlled by a PID controller.

PWM code examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

PWM initialization examples:

```
// Setup a PWM output with precise current-measurement
IO_PWM_Init( IO_PWM_00
            , 100                // frequency is 100 Hz
            , TRUE               // positive polarity
            , TRUE               // diag margin
            , 1500               // signal overload above 1.5A
            , NULL );            // safety configuration; currently not
                                // supported

IO_POWER_Set( IO_INT_POWERSTAGE_ENABLE, IO_POWER_ON ); // enable
high-side outputs
```

PWM task examples:

```
ubyte2 curr;
bool new;

IO_PWM_SetDuty( IO_PWM_00
                , 0x8000           // set duty cycle to 50%
                , NULL );          // duty-cycle feedback is ignored

IO_PWM_GetCur( IO_PWM_00         // read current value of PWM output
               , &curr            // variable to store current
               , &new );          // variable to store information if a
current value is available
```

Current-controlled PWM code examples

Current-controlled PWM initialization examples:

```
// PID controller configuration
IO_PID_CONFIG pid_cfg =
{
    .Kd = 500U,                // Differential gain scaled by 10000 =>
    gain of 0.05
    .Kff = 10000U,             // Feed-forward gain scaled by 1000 =>
    gain of 10
    .Ki = 15000U,              // Integral gain scaled by 10000 =>
    gain of 15
    .Kp = 500U,                // Proportional gain scaled by 1000 =>
    gain of 0.05
    .max_limit = 65535U,       // Maximal allowed duty cycle is 100%
    (2^16)
    .min_limit = 0U,           // Minimal allowed duty cycle is 0% (0)
};

// Setup a current-controlled PWM output
IO_PWM_CurrentInit( IO_PWM_00
                    , 100      // frequency is 100 Hz
                    , &pid_cfg // use custom PID configuration
                    , 10       // use 10ms as cycle time for the
    PID controller
                    , NULL     // Handle for PID controller not
    needed => pass NULL
                    , NULL );  // safety configuration; currently
    not supported

IO_POWER_Set( IO_INT_POWERSTAGE_ENABLE, IO_POWER_ON ); // enable
    high-side outputs
```

PWM task examples:

```
ubyte2 curr;
bool new;

IO_PWM_SetCur ( IO_PWM_00
                , 500      // set current to 500mA
                , NULL );  // duty-cycle feedback is ignored

IO_PWM_GetCur( IO_PWM_00 // read current value of PWM output
                , &curr    // variable to store current
                , &new );  // variable to store information if a
    current value is available
```

Macro Definition Documentation

#define IO_PWM_CURRENT_QUEUE_MAX 6U

maximum number of items in the current queue

Typedef Documentation

typedef struct _io_pwm_current_queue IO_PWM_CURRENT_QUEUE

PWM current measurement queue.

Stores results of the equidistant current measurement.

The queue holds all current measurement since the last retrieval via the step function

`IO_PWM_GetCur()`.

typedef struct _io_pwm_current_safety_conf IO_PWM_CURRENT_SAFETY_CONF

Safety configuration for the PWM outputs.

Stores all relevant safety configuration parameters for the PWM outputs.

Attention

For the current check to work properly it is necessary that the application does not change the set-point of the PID controller before the dead time elapses! for example if a dead time of 10ms is set, the application shall change the set-point for the current value less frequently than 10ms!

To disable the current check, set the value of the parameters `current_tolerance` to 65535 and `dead_time` to 4294967295.

typedef struct _io_pwm_safety_conf IO_PWM_SAFETY_CONF

Safety configuration for the PWM outputs.

Stores all relevant safety configuration parameters for the PWM outputs.

Function Documentation

IO_ErrorType IO_PWM_CurrentDelnit (**IO_PIN** **pwm_channel**)

Deinitializes a current-controlled PWM output. Allows the re-initialization of the output by other functions.

Parameters

pwm_channel PWM channel (**IO_PWM_00** .. **IO_PWM_05**)

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_INVALID_CHANNEL_ID	the given channel id does not exist
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CH_CAPABILITY	The given channel has no PWM capability

IO_ErrorType

IO_PWM_CurrentInit (IO_PIN	pwm_channel,
ubyte2	frequency,
const IO_PID_CONFIG *const	pid_config,
ubyte1	pwm_period_multi,
ubyte1 *const	pid_handle,
const IO_PWM_CURRENT_SAFETY_CONF *const	safety_conf
)	

Setup a current controlled PWM output.

Parameters

pwm_channel	PWM channel (IO_PWM_00 .. IO_PWM_05)
frequency	PWM frequency (15Hz .. 250Hz, only frequencies with a period of an integral multiple of 1ms are possible)
pid_config	Coefficients (Ki, Kd, Kp) for PID controller
pwm_period_multi	Period multiplier for PID controller. The cycle time for the PID controller is calculated by multiples of the PWM period time. <ul style="list-style-type: none">• Range: ((1 .. 255) / PWM_period_[ms])
pid_handle	Contains handle of the PID controller used for this output. Can be used to interact with the PID controller after initializing it (e.g. calling IO_PID_SetIntegrator)
safety_conf	Safety configuration for PWM output

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_CHANNEL_BUSY	the PWM output channel or the timer input channel is currently used by another function
IO_E_INVALID_CHANNEL_ID	the PWM output channel or the timer input channel id don't exist
IO_E_INVALID_PARAMETER	a given parameter is out of range
IO_E_CH_CAPABILITY	The PWM capability of this channel has not been activated.
IO_E_DRIVER_NOT_INITIALIZED	The common driver init function has not been called before
IO_E_SW_INTERNAL	Internal SW error when handling configuration tables (index out of range)
IO_E_GROUP_CONFLICT	configuring the output not allowed due to conflicts with other in/outputs in the same group
IO_E_INVALID_SAFETY_CONFIG	one or more parameters of the safety configuration are invalid (out of range)
IO_E_SAFETY_NOT_SUPPORTED	the given channel does not support safety features
IO_E_DRV_SAFETY_CONF_NOT_CONFIG	the driver has not been initialized as safety device - therefore the safety feature is not available for this channel
IO_E_TASK_NO_FREE_SLOTS	No more free slots to setup task function

Remarks

- The product of (PWM_period_[ms] x multiplier) must not exceed 255! So for a frequency of 250Hz the maximum multiplier value equals 63. For a frequency of 15Hz (15.625Hz, 64ms period time) the maximum multiplier value equals 3.
- The associated timer loopback channel will also be configured for open load and short circuit detection for all channels.
- The associated analog feedback channel will also be configured for all channels.
- The associated precise current-measurement will be configured for the channels **IO_PWM_00** .. **IO_PWM_05**. These PWM outputs will be switched off after 1 second if the the continuous current is above 3.0A but below 4.0A or switched off immediately if the current exceeds 4.0A. After a recovery time of 1 second the output stages get enabled again.
- **IO_PWM_00** and **IO_PWM_01** have their own frequency time base (variable frequency)
- **IO_PWM_02**, **IO_PWM_03**, **IO_PWM_04** and **IO_PWM_05** share one frequency time base. It's only allowed to configure these outputs with the same PWM frequency.

Note

When one PWM channel of a frequency group is initialized, it's only allowed to initialize the other channels of this group with the same frequency. Otherwise the function will return **IO_E_GROUP_CONFLICT**.

Attention

The parameter `pwm_period_multi` specifies the cycle time in multiples of the period time of the PWM signal. However for backwards compatibility the cycle time of the PID controller can not exceed 255ms. Therefore the upper limit of this parameter depends on the set PWM frequency. For example for a 100Hz PWM signal (10ms period time) the highest number for the period multiplier of the PID controller is 25.

`IO_ErrorType IO_PWM_DeInit (IO_PIN pwm_channel)`

Deinitializes a PWM output. Allows the re-initialization of the output by other functions.

Parameters

pwm_channel PWM channel (`IO_PWM_00 .. IO_PWM_05, IO_PWM_10 .. IO_PWM_11`)

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	the given channel id does not exist
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	the given channel is not configured
<code>IO_E_CH_CAPABILITY</code>	The given channel has no PWM capability

Remarks

- If a current measurement is configured for the given PWM channel, the current measurement channel will also be deinitialized.

`IO_ErrorType IO_PWM_GetCur (IO_PIN pwm_channel, ubyte2 *const current, bool *const fresh)`

Returns the measured current of the given channel.

Parameters

pwm_channel PWM channel, one of

- `IO_PWM_00 .. IO_PWM_05`

current Measured current Range: 0..7575 (0mA..7575mA)

fresh Indicates if new values are available since the last call.

- `TRUE`: Value in "current" is valid
- `FALSE`: No new value available.

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	Everything fine
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	The given PWM channel is not configured
<code>IO_E_INVALID_CHANNEL_ID</code>	The given channel id does not exist

IO_E_ADC_INVALID	An ADC error occurred
IO_E_PWM_CURRENT_INACCURATE	Current measurement has reduced accuracy
IO_E_CH_CAPABILITY	The given channel is not a PWM channel or the given PWM channel has no current measurement
IO_E_NULL_POINTER	A NULL pointer has been passed to the function

Attention

The current measurement for all PWM channels is equidistant, which means that the sampling happens synchronous to the PWM period. Every 1ms a current value will be captured. The captured current values will be averaged over the time of one period of the PWM signal and then provided to the application.

Only PWM channels `IO_PWM_00` .. `IO_PWM_05` have current-measurement.

Remarks

- When the function `IO_PWM_GetCur()` is called, the internal queue holding the values of the current measurement is flushed. If the function is called more than once in a cycle it may or may not deliver new values, depending on how many values the equidistant current measurement has sampled since the last call.
- If there is no new current value available (for example the function `IO_PWM_GetCur()` gets called more frequently than the PWM period) the flag `fresh` will be set to `FALSE`.
- If the functions `IO_PWM_GetCurQueue()` `IO_PWM_GetCur()` are called after another, only the first function will deliver a current value because both functions will empty the internal measurement queue.

IO_ErrorType

```
IO_PWM_GetCurQueue ( IO_PIN pwm_channel,
                    IO_PWM_CURRENT_QUEUE *const current_queue
                    )
```

Returns the measured current values since the last call of the given channel.

Parameters

pwm_channel PWM channel, one of

- `IO_PWM_00` .. `IO_PWM_05`
- `IO_PWM_10` .. `IO_PWM_11`

current_queue Queue holding the current values since the last call of the step function including a queue overrun flag and a counter for the quantity of available values

Returns

`IO_ErrorType`

Return values

IO_E_OK	Everything fine
IO_E_CHANNEL_NOT_CONFIGURED	The given PWM channel is not configured
IO_E_INVALID_CHANNEL_ID	The given channel id does not exist
IO_E_ADC_INVALID	An ADC error occurred
IO_E_PWM_CURRENT_INACCURATE	Current measurement has reduced accuracy

IO_E_CH_CAPABILITY

The given channel is not a PWM channel or the given PWM channel has no current measurement

IO_E_NULL_POINTER

A NULL pointer has been passed to the function

Attention

The current measurement for all PWM channels is equidistant which means that the sampling happens synchronous to the PWM period. Every 1ms a current value will be captured. The captured current values will be averaged over the time of one period of the PWM signal and then provided to the application in a queue. The queue storage was chosen to avoid the loss of any measurement value if the user application runs asynchronous to the current measurement or if its cycle time is lower than the PWM period. (Size of queue: `IO_PWM_CURRENT_QUEUE_MAX`, Queue data type: `IO_PWM_CURRENT_QUEUE`)

Only PWM channels `IO_PWM_00` .. `IO_PWM_05` have current-measurement.

Remarks

- When the function `IO_PWM_GetCurQueue()` is called, the internal queue holding the values of the current measurement is flushed. If the function is called more than once in a cycle it may or may not deliver new values, depending on how many values the equidistant current measurement has sampled since the last call.
- If there is no new current value available (for example the function `IO_PWM_GetCurQueue()` gets called more frequently than the PWM period) the value `count` in the structure `current_queue` will be 0.
- If the functions `IO_PWM_GetCurQueue()` and `IO_PWM_GetCur()` are called after another, only the first function will deliver a current value because both functions will empty the internal measurement queue.

```
IO_ErrorType IO_PWM_Init ( IO_PIN
                           ubyte2
                           bool
                           bool
                           ubyte2
                           const IO_PWM_SAFETY_CONF *const
                           pwm_channel,
                           frequency,
                           polarity,
                           diag_margin,
                           overload_limit,
                           safety_conf
                           )
```

Setup single PWM output.

Parameters

- pwm_channel** PWM channel (`IO_PWM_00` .. `IO_PWM_05` Or `IO_PWM_10` .. `IO_PWM_11`)
- frequency** PWM frequency (15Hz .. 1000Hz, only frequencies with a period of an integral multiple of 1ms are possible)
- polarity** Polarity of output signal
- `FALSE`: Low output signal is variable
 - `TRUE`: High output signal is variable
- diag_margin** Indicate if a margin should be applied or not.
- `TRUE`: margin is on
 - `FALSE`: no margin will be applied
- overload_limit** Limit in mA above which a `IO_E_PROT_USER_OVERLOAD` error will be reported

- overload_limit configured to 0: overload limit disabled;
IO_E_PROT_USER_OVERLOAD error will not be reported
- overload_limit configured to 1mA .. 2999mA:
IO_E_PROT_USER_OVERLOAD error will be reported via function **IO_PWM_SetDuty()** or **IO_PWM_SetCur()** if the measured current is above defined overload_limit
- if measured current exceeds 3000mA other error codes than **IO_E_PROT_USER_OVERLOAD** are reported (see error codes of function **IO_PWM_SetDuty()** or **IO_PWM_SetCur()**):
IO_E_PROT_TEMP_OVERLOAD, **IO_E_PROT_ACTIVE** and **IO_E_PROT_FATAL**)

safety_conf Safety configuration.

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_BUSY

the PWM output channel or the timer input channel is currently used by another function

IO_E_INVALID_CHANNEL_ID

the PWM output channel or the timer input channel id don't exist

IO_E_INVALID_PARAMETER

a given parameter is out of range

IO_E_CH_CAPABILITY

The PWM capability of this channel has not been activated.

IO_E_DRIVER_NOT_INITIALIZED

The common driver init function has not been called before

IO_E_SW_INTERNAL

Internal SW error when handling configuration tables (index out of range)

IO_E_GROUP_CONFLICT

configuring the output not allowed due to conflicts with other in/outputs in the same group

IO_E_INVALID_SAFETY_CONFIG

one or more parameters of the safety configuration are invalid (out of range)

IO_E_SAFETY_NOT_SUPPORTED

the given channel does not support safety features

IO_E_DRV_SAFETY_CONF_NOT_CONFIG the driver has not been initialized as safety device - therefore the safety feature is not available for this channel

IO_E_TASK_NO_FREE_SLOTS

No more free slots to setup task function

Remarks

- For the equidistant current measurement to work properly only frequencies that have a period which is an integral multiple of 1ms are allowed. Periods which are not a integral multiple of 1ms will be rounded down to the next lower integral multiple of 1ms which means that the frequency will be calculated with the next lower integral period. For example a wanted frequency of 180Hz means 5.55ms. The next lower integral value is 5ms therefore the resulting frequency is 200Hz in this case. Possible frequencies are:

Available frequencies:

Period(ms)	Frequency(Hz)
1	1000
2	500
3	333 (333.333333333333)
4	250
5	200
6	166 (166.666666666667)
7	142 (142.857142857143)
8	125
9	111 (111.111111111111)
10	100
11	90 (90.9090909090909)
12	83 (83.3333333333333)
13	76 (76.9230769230769)
14	71 (71.4285714285714)
15	66 (66.6666666666667)
16	62 (62.5)
17	58 (58.8235294117647)
18	55 (55.5555555555556)
19	52 (52.6315789473684)
20	50
21	47 (47.61904762)
22	45 (45.45454545)
23	43 (43.47826087)
24	41 (41.66666667)
25	40
26	38 (38.46153846)
27	37 (37.03703704)
28	35 (35.71428571)
29	34 (34.48275862)
30	33 (33.33333333)
31	32 (32.25806452)
32	31 (31.25)
33	30 (30.3030303)
34	29 (29.41176471)
35	28 (28.57142857)
37	27 (27.02702703)
38	26 (26.31578947)
40	25
41	24 (24.3902439)
43	23 (23.25581395)
45	22 (22.22222222)

47	21 (21.27659574)
50	20
52	19 (19.23076923)
55	18 (18.18181818)
58	17 (17.24137931)
62	16 (16.12903226)
64	15 (15.625)

Remarks

- The associated timer loopback channel will also be configured for open load and short circuit detection for all channels.
- The associated analog feedback channel will also be configured for all channels.
- The associated precise current-measurement will be configured for the channels `IO_PWM_00` .. `IO_PWM_05`. These PWM outputs will be switched off after 1 second if the the continuous current is above 3.0A but below 4.0A or switched off immediately if the current exceeds 4.0A. After a recovery time of 1 second the output stages get enabled again.
- The associated overcurrent-measurement will be configured for the channels `IO_PWM_10` .. `IO_PWM_11`. These PWM outputs will be switched off after 1 seconds if the the continuous current is above 3.0A but below 4.0A or switched off immediately if the current exceeds 4.0A. After a recovery time of 1 second the output stages get enabled again.
- The duty cycle cannot exceed the margin of 100us(lower boundary) and 250us(upper boundary) used for diagnostic if the parameter `diag_margin` is `TRUE`. This mode is important for hydraulic coils If the parameter `diag_margin` is `FALSE`, no duty cycle range margin will be applied
- `IO_PWM_00`, `IO_PWM_01`, `IO_PWM_10` and `IO_PWM_11` have their own frequency time base (variable frequency)
- `IO_PWM_02`, `IO_PWM_03`, `IO_PWM_04` and `IO_PWM_05` share one frequency time base. It's only allowed to configure these outputs with the same PWM frequency.
- Static friction and stiction can cause a hysteresis and make the control of a hydraulic valve erratic and unpredictable. In order to counteract these hysteretic effects, small vibrations about the desired position shall be created in the valve. This constantly breaks the static friction ensuring that it will move even with small input changes, and the effects of hysteresis are average out. A proper setting of PWM frequency according to the resonance frequency of the actuator allows to adjust this desired small vibration, low enough in amplitude to prevent noticeable oscillations on the hydraulic output but sufficient high to prevent friction. The PWM frequency can be set in the range of 15 .. 1000Hz, a typical range for hydraulic valves to operate without friction is 90 .. 160Hz.

Note

- If a PWM channel is initialized as safety and a valid safety configuration is given, the diagnostic margin must be set to `TRUE`. Otherwise the initialization will fail and the error code `IO_E_INVALID_PARAMETER` is returned.
- When one PWM channel of a frequency group is initialized, it's only allowed to initialize the other channels of this group with the same frequency. Otherwise the function will return `IO_E_GROUP_CONFLICT`.

Attention

- For the current measurement circuitry of the channels `IO_PWM_00 .. IO_PWM_05` a offset compensation will be made at startup of the channel. The application software has to make sure that no current is driven by the actuator coil (through the free wheeling diode) during this time. Otherwise the offset compensation will be wrong and the channel startup may fail. That means that in case a PWM channel with current measurement is re-initialized during runtime, the application has to make sure that there is enough time between the calls of `IO_PWM_DeInit` and `IO_PWM_Init` or `IO_PWM_CurrentInit` and `IO_PWM_CurrentDeInit` for the actuator coil to finish its recovery phase (time depends on inductance of the load). This is mainly relevant for channels that will be re-initialized during runtime and are connected to a inductive load.

```
IO_ErrorType IO_PWM_SetCur ( IO_PIN          pwm_channel,
                             ubyte2          current,
                             ubyte4 *const    duty_cycle_fb
                             )
```

Sets the current for a current-controlled PWM channel.

Parameters

pwm_channel PWM channel, one of

- `IO_PWM_00 .. IO_PWM_05`

current Current in mA which shall be output

duty_cycle_fb Duty cycle feedback for the channels (optional): high-time in us

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything fine

`IO_E_CHANNEL_NOT_CONFIGURED`

the given channel is not configured

`IO_E_INVALID_CHANNEL_ID`

the given channel id does not exist

`IO_E_CH_CAPABILITY`

The given channel has no PWM capability

`IO_E_PWM_OPEN_LOAD`

open load has been detected

`IO_E_PWM_SHORT_CIRCUIT`

short circuit to GND has been detected

`IO_E_PWM_SHORT_BATTERY`

short to UBAT has been detected

`IO_E_PWM_OUTPUT_DISABLED`

The PWM output has been disabled by the main CPU and there was no output signal detected on the output.

`IO_E_PWM_CHANNEL_STARTUP`

the PWM output is in it's startup phase

`IO_E_PWM_OUTPUT_STARTUP_ERROR`

an error occurred on the channel mode

`IO_E_ADC_INVALID`

A ADC error occurred

`IO_E_SW_INTERNAL`

Internal SW error when handling configuration tables (index out of range)

`IO_E_PWM_CAPTURE_ERROR`

A capture error occurred on the loopback channel. This error can occur if two edges of the feedback signal are too close to each other and the internal timer can not measure the time difference anymore (for example spikes)

IO_E_PROT_USER_OVERLOAD

caused by the switching of inductive loads in electric motors).

Output current is above the threshold configured on initialization. The output remains active.

IO_E_PROT_TEMP_OVERLOAD

Output current is above 3.0A and output will temporarily be switched off if the current does not decrease to or below 3.0A within the next 1 seconds.

IO_E_PROT_ACTIVE

Output is disabled, protection is active because of too high output current (over 4A). The driver will try to re-enable the output again in 1 second

IO_E_PROT_FATAL

Output is disabled, protection is active because of too high output current (over 6A). The driver will try to re-enable the output again in 1 second

IO_E_PROT_REENABLE

The IO-Driver signals that the switched off PWM output is going to be re-enabled.

```
IO_ErrorType IO_PWM_SetDuty ( IO_PIN
                             ubyte2 pwm_channel,
                             ubyte4 duty_cycle,
                             *const duty_cycle_fb
                             )
```

Set the duty cycle for a PWM channel.

Parameters

pwm_channel PWM channel, one of

- **IO_PWM_00** .. **IO_PWM_05**
- **IO_PWM_10** .. **IO_PWM_11**

duty_cycle Duty cycle for the channel (0..65535)

duty_cycle_fb Duty cycle feedback for the channels (optional): high-time in us

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_NOT_CONFIGURED

the given channel is not configured

IO_E_INVALID_CHANNEL_ID

the given channel id does not exist

IO_E_CH_CAPABILITY

The given channel has no PWM capability

IO_E_PWM_OPEN_LOAD_OR_SHORT_BATTERY

an open-load or short-to-battery condition has been detected (at the point in time of the occurrence of this error it is not yet possible to

IO_E_PWM_OPEN_LOAD
IO_E_PWM_SHORT_CIRCUIT

IO_E_PWM_SHORT_BATTERY
IO_E_PWM_OUTPUT_DISABLED

IO_E_PWM_CHANNEL_STARTUP
IO_E_PWM_OUTPUT_STARTUP_ERROR

IO_E_ADC_INVALID
IO_E_SW_INTERNAL

IO_E_PWM_CAPTURE_ERROR

IO_E_PROT_USER_OVERLOAD

IO_E_PROT_TEMP_OVERLOAD

IO_E_PROT_ACTIVE

IO_E_PROT_FATAL

IO_E_PROT_REENABLE

distinguish between open-load and short-to-battery conditions)
open load has been detected
short circuit to GND has been detected
detected
short to UBAT has been detected
The PWM output has been disabled by the main CPU and there was no output signal detected on the output.
the PWM output is in it's startup phase
an error occurred on the channel mode
A ADC error occurred
Internal SW error when handling configuration tables (index out of range)
A capture error occurred on the loopback channel. This error can occur if two edges of the feedback signal are too close to each other and the internal timer can not measure the time difference anymore (for example spikes caused by the switching of inductive loads in electric motors).
Output current is above the threshold configured on initialization. The output remains active.
Output current is above 3.0A and output will temporarily be switched off if the current does not decrease to or below 3.0A within the next 1 seconds (only [IO_PWM_00..IO_PWM_05](#))
Output is disabled, protection is active because of too high output current (over 4A). The driver will try to re-enable the output again in 1 second
Output is disabled, protection is active because of too high output current (over 6A). The driver will try to re-enable the output again in 1 second
The IO-Driver signals that the switched off PWM output is going to be re-enabled.

Remarks

- The duty cycle cannot exceed a margin of 100us(lower boundary) and 250us(upper boundary) used for diagnostic if the parameter `diag_margin` was set TRUE (via the function `IO_PWM_Init()`). This mode is important for hydraulic coils. If the parameter `diag_margin` is FALSE, no duty cycle range margin will be applied.
- If the parameter `duty_cycle_fb` is NULL, the parameter is ignored. The parameter `duty_cycle_fb` returns the measured pulse-width of the PWM signal in the last round in us. If the duty cycle measurement is not finished yet, the parameter `duty_cycle_fb` holds the value 0.

Note

- The PWM outputs must be enabled with `IO_POWER_Set`. Otherwise the outputs remain disabled.
- PWM outputs configured with frequencies above 250Hz can only sustain a continuous current of 100mA.



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

Functions

IO_RTC.h File Reference

RTC functions, provides exact timing functions. [More...](#)

```
#include "IO_Driver.h"
```

Functions

IO_ErrorType **IO_RTC_Init** (void)
Initializes the RTC clock.

IO_ErrorType **IO_RTC_StartTime** (ubyte4 *const timestamp)
Returns an RTC timestamp.

ubyte4 **IO_RTC_GetTimeUS** (ubyte4 timestamp)
Returns the passed time.

IO_ErrorType **IO_RTC_PeriodicInit** (ubyte2 period, rtc_eventhandler_ptr event_handler)
Initializes the Periodic Timer.

IO_ErrorType **IO_RTC_PeriodicDeInit** (void)

Deinitializes the Periodic Timer.

Detailed Description

RTC functions, provides exact timing functions.

Provides setup and utility functions for the Real Time Clock. The RTC is used for SW timings.

The function `IO_RTC_StartTime()` returns a timestamp. The function `IO_RTC_GetTimeUS()` returns the time which has passed since the timestamp-value passed as an argument to this function. The application can use these two functions as often as it needs to. For different timing tasks only different timestamp variables need to be used.

code example:

The example initializes the RTC driver, and implements a loop which is executed every 5ms:

```
ubyte4 time_stamp;

// RTC can be initialized:
// by initializing the driver
IO_Driver_Init ( IO_DRIVER_MODE_DEFAULT, NULL
                );

// or by only initializing the RTC module
// IO_RTC_Init();

while (1)
{
    IO_RTC_StartTime (&time_stamp);
    // start time (get timestamp)

    task();          // user task function

    while (IO_RTC_GetTimeUS (time_stamp) <
           5000);    // wait until 5ms have passed
}
```

Function Documentation

ubyte4 `IO_RTC_GetTimeUS (ubyte4 timestamp)`

Returns the passed time.

The function returns the time in us which has passed since the given timestamp has been taken (via the function

`IO_RTC_StartTime()`)

Parameters

timestamp Timestamp received from a call of
`IO_RTC_StartTime()`

Returns

ubyte4

Remarks

- If the RTC module has not initialized, the function will return 0
- Please keep in mind that the time between `IO_RTC_StartTime()` and `IO_RTC_GetTimeUS()` for one timestamp should not exceed 74min (overflow)

IO_ErrorType `IO_RTC_Init (void)`

Initializes the RTC clock.

Initializes the RTC clock to a $f_{sys} / 80$. For a system clock of 80MHz the RTC resolution is 1us

Returns

IO_ErrorType

Return values

IO_E_OK everything fine
IO_E_CHANNEL_BUSY the module has been initialized before

Remarks

Module is initialized only once.

- The RTC driver is initialized when the `IO_Driver_Init()` function is called. Therefore it will return `IO_E_CHANNEL_BUSY` if it is called after this function. This means that `IO_RTC_Init()` needs to be called only when `IO_Driver_Init()` is not used in the respective application.

IO_ErrorType `IO_RTC_PeriodicDeInit (void)`

Deinitializes the Periodic Timer.

Deinitializes a Periodic Timer and stops it

Returns

IO_ErrorType

Return values

IO_E_OK everything fine
IO_E_PERIODIC_NOT_CONFIGURED the channel has not yet been initialized

Remarks

- This function can also be used without initializing the RTC driver

IO_ErrorType

```
IO_RTC_PeriodicInit ( ubyte2          period,
                      rtc_eventhandler_ptr event_handler
                      )
```

Initializes the Periodic Timer.

Initializes a Periodic Timer

Parameters

period Period on which the event handler should be called. unit: us (500..65535)

event_handler Function pointer to the periodic event handler

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_NULL_POINTER	a null pointer has been passed
IO_E_INVALID_PARAMETER	a invalid parameter has been passed
IO_E_BUSY	the channel has been initialized before

Remarks

- This function can also be used without initializing the RTC driver

IO_ErrorType

IO_RTC_StartTime

(**ubyte4** *const **timestamp**)

Returns an RTC timestamp.

Returns a timestamp which can be used for RTC timing functions

Parameters

timestamp Pointer for the returned timestamp value

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_NULL_POINTER

a null pointer has been passed

IO_E_CHANNEL_NOT_CONFIGURED the module has not been initialized



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page		Related Pages	Data Structures	Files
File List	Globals			
inc				

Functions

IO_UART.h File Reference

IO Driver functions for UART communication. [More...](#)

```
#include "IO_Driver.h"
```

Functions

IO_ErrorType **IO_UART_Init** (**IO_PIN** channel, **ubyte4** baudrate)
Initialization of the UART communication driver.

IO_ErrorType **IO_UART_DeInit** (**IO_PIN** channel)
Deinitialization of the UART channel.

IO_ErrorType **IO_UART_Task** (void)
Task function for UART communication.

IO_ErrorType **IO_UART_Read** (**IO_PIN** channel, **ubyte1** *const data, **ubyte1** len, **ubyte1** *const rx_len)
Read data from serial interface.

IO_ErrorType IO_UART_Write (**IO_PIN** channel, const **ubyte1** *const data, **ubyte1** len, **ubyte1** *const tx_len)
Write data to serial interface.

IO_ErrorType IO_UART_GetRxStatus (**IO_PIN** channel, **ubyte1** *const rx_len)
Retrieve the status of the receive buffer.

IO_ErrorType IO_UART_GetTxStatus (**IO_PIN** channel, **ubyte1** *const tx_len)
Retrieve the status of the transmit buffer.

Detailed Description

IO Driver functions for UART communication.

The Universal Serial Interface Channel module (USIC) is a flexible interface module covering several serial communication protocols.

A USIC module contains two independent communication channels named UxC0 and UxC1, with x being the number of the USIC module.

The UART communication driver uses the USIC module as universal asynchronous receiver transmitter.

When initializing the UART communication driver it is possible to define the baudrate individually. Apart from the baudrate, the following parameters are fixed:

- databits: 8
- stopbits: 1
- parity: none

The UART buffers

The XC2000 has 16 byte transmit and receive buffers for every UART channel. The UART driver additionally implements a SW buffer of 128 bytes.

This means that a maximum of 128 bytes can be written, using the `IO_UART_Write()` function. The UART Task function sequentially copies the data to the HW buffer and sends it. The function `IO_UART_GetTxStatus()` returns the number of remaining bytes in the SW buffer. The write function can be called any time, as long as the SW buffer is not full.

The UART Task function is called automatically every SW cycle by `IO_Driver_TaskEnd()`.

To speed up the transmission the application can call the `IO_UART_Task()` function at any time.

In every software cycle a maximum of 16 bytes (size of HW buffer) can be received via a UART interface.

The data is copied to the SW buffer by the UART task function. To increase the number of bytes which can be received within a cycle the application can call the function `IO_UART_Task()` during the cycle.

The function `IO_UART_GetRxStatus()` returns the number of bytes in the SW buffer which is available for reading.

UART code examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

UART initialization example:

```
// setup UART channel:
IO_UART_Init( IO_UART
              , 115200 );           // 115200
                                   baud
```

UART read example:

```
ubyte1 data[40] = {0};
ubyte1 size;

// check if new bytes have been received
IO_UART_GetRxStatus(IO_UART, &size);
if (size > 0)
{
    // read a maximum of 40 bytes
    IO_UART_Read(IO_UART, data, 40, &size);

    // data now holds the received data
    // size holds the number of actually read
    bytes.
}
```

UART write example:

```
ubyte1 data[5] = {0, 1, 2, 3, 4};
ubyte1 size;

// write data to UART buffer:
```

```
IO_UART_Write(IO_UART, data, 5, &size);  
// size holds the number of actually written  
    bytes.  
  
// check if data has been transmitted  
IO_UART_GetTxStatus(IO_UART, &size);  
// when size returns 0 all the data has been  
    transmitted
```

Function Documentation

IO_ErrorType IO_UART_DeInit (IO_PIN channel)

Deinitialization of the UART channel.

Allows re-initialization by `IO_UART_Init()`

Parameters

channel UART Channel: `IO_UART`

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything fine

`IO_E_INVALID_CHANNEL_ID`

channel number is invalid

`IO_E_CHANNEL_NOT_CONFIGURED`

channel has not been initialized

`IO_E_CH_CAPABILITY`

channel should not be capable of the desired functionality

IO_ErrorType

IO_UART_GetRxStatus

(**IO_PIN** channel,
 ubyte1 *const rx_len
)

Retrieve the status of the receive buffer.

Returns the current status of the receive buffer: new data available, error has occurred...

Parameters

channel UART Channel: `IO_UART`

rx_len Number of received data frames in receive buffer

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything fine

`IO_E_UART_BUFFER_FULL`

SW receive queue is full and data has been lost

`IO_E_UART_OVERFLOW`

HW receive buffer overrun

`IO_E_UART_PARITY`

parity check failed

`IO_E_INVALID_CHANNEL_ID`

invalid channel ID has been passed

`IO_E_NULL_POINTER`

null pointer has been passed

`IO_E_CHANNEL_NOT_CONFIGURED`

channel has not been initialized

`IO_E_CH_CAPABILITY`

channel is not capable of the desired functionality

`IO_ErrorType`

(`IO_PIN`

channel,

IO_UART_GetTxStatus

```
ubyte1 *const tx_len  
)
```

Retrieve the status of the transmit buffer.

Returns the number of remaining bytes in the SW buffer.

Parameters

channel UART Channel: `IO_UART`

tx_len Number of received data frames in receive buffer

Returns

`IO_ErrorType`

Return values

`IO_E_OK`

everything fine

`IO_E_INVALID_CHANNEL_ID`

invalid channel ID
has been passed

`IO_E_NULL_POINTER`

null pointer has
been passed

`IO_E_CHANNEL_NOT_CONFIGURED`

channel has not
been initialized

`IO_E_CH_CAPABILITY`

channel is not
capable of the
desired
functionality

```
IO_ErrorType IO_UART_Init ( IO_PIN channel,  
                             ubyte4 baudrate  
                             )
```

Initialization of the UART communication driver.

Initialization of UART Serial Communication Driver

- Enables module
- Configures module for ASC
- Initializes SW queue
- Sets the communication mode to 8N1

Parameters

channel UART Channel: `IO_UART`

baudrate Baud rate in baud/s (1200 ... 115200)

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_INVALID_CHANNEL_ID</code>	channel number is invalid
<code>IO_E_INVALID_PARAMETER</code>	a given parameter is out of range
<code>IO_E_CHANNEL_BUSY</code>	the channel is already initialized
<code>IO_E_CH_CAPABILITY</code>	channel is not capable of the desired functionality

Remarks

- Module is initialized only once. To re-initialize the module, the function `IO_UART_DeInit()` needs to be called.
- The UART channel is not available on the external connector but only on the debugging interface.

```
IO_ErrorType IO_UART_Read ( IO_PIN          channel,
                             ubyte1 *const data,
                             ubyte1          len,
                             ubyte1 *const rx_len
                             )
```

Read data from serial interface.

Reads the data from the SW buffer. If new data is available the content will be transferred from the SW buffer to the data array. In case there is no new data available the array is untouched.

Parameters

channel UART Channel: `IO_UART`
data Data array
len Maximum size of data array
rx_len Actually read bytes

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_UART_BUFFER_FULL</code>	SW receive queue is full and data has been lost
<code>IO_E_UART_OVERFLOW</code>	HW receive buffer overrun
<code>IO_E_UART_PARITY</code>	parity check failed
<code>IO_E_INVALID_CHANNEL_ID</code>	invalid channel ID has been passed
<code>IO_E_NULL_POINTER</code>	null pointer has been passed

IO_E_CHANNEL_NOT_CONFIGURED	channel has not been initialized
IO_E_CH_CAPABILITY	channel is not capable of the desired functionality

IO_ErrorType IO_UART_Task (void)

Task function for UART communication.

The task function copies the data to be sent from the SW buffer to the HW buffer and the received data from the HW buffer to the SW buffer.

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_UART_BUFFER_FULL	SW receive queue is full and data has been lost
IO_E_UART_OVERFLOW	HW receive buffer overrun
IO_E_UART_PARITY	parity check failed

IO_ErrorType

```
IO_UART_Write ( IO_PIN channel,
const ubyte1 *const data,
ubyte1 len,
ubyte1 *const tx_len
)
```

Write data to serial interface.

Writes the data to the SW buffer and starts the transmission.

Parameters

channel UART Channel: `IO_UART`
data Data array
len Number of bytes in data array
tx_len Actually written bytes

Returns

`IO_ErrorType`

Return values

<code>IO_E_OK</code>	everything fine
<code>IO_E_UART_BUFFER_FULL</code>	SW transmit queue is full, no data has been written
<code>IO_E_INVALID_CHANNEL_ID</code>	invalid channel ID has been passed
<code>IO_E_NULL_POINTER</code>	null pointer has been passed
<code>IO_E_CHANNEL_NOT_CONFIGURED</code>	channel has not been initialized
<code>IO_E_CH_CAPABILITY</code>	channel is not capable of the desired functionality

Note

It is highly recommended not to modify the content of the data array while a write operation is ongoing! Furthermore

please note that the data array must not be a local variable since the driver does not create a copy of the content but saves the address of the pointer.



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

Functions

IO_Vout.h File Reference

IO Driver functions for voltage outputs. [More...](#)

```
#include "IO_Driver.h" #include "IO_PID.h"
```

Functions

IO_ErrorType **IO_VOut_Init** (**IO_PIN** vout_channel, const **IO_PID_CONFIG** *const pid_config, **ubyte1** *const pid_handle)
Setup one voltage output channel.

IO_ErrorType **IO_VOut_DeInit** (**IO_PIN** vout_channel)
Deinitializes one voltage output channel.

IO_ErrorType **IO_VOut_SetVoltage** (**IO_PIN** vout_channel, **ubyte2** output_voltage, **ubyte2** *const voltage_fb)
Sets the output voltage of one VOut channel.

Detailed Description

IO Driver functions for voltage outputs.

Contains all service functions for the voltage outputs. Up to 6 channels can be configured: `IO_VOUT_00 .. IO_VOUT_05`

- The pins for PVG- and Voltage-Outputs are organized in two groups of three
 - Group 1: `IO_PVG_00/IO_VOUT_00`, `IO_PVG_01/IO_VOUT_01` and `IO_PVG_02/IO_VOUT_02`
 - Group 2: `IO_PVG_03/IO_VOUT_03`, `IO_PVG_04/IO_VOUT_04` and `IO_PVG_05/IO_VOUT_05` Whenever a pin within a group is configured either as PVG or voltage output, all other pins within the same group must be either remain unconfigured, or be configured with the same type. For example, if `IO_PIN_J2` (`IO_PVG_01/IO_VOUT_01`) is configured as voltage output, pins `IO_PIN_K2` (`IO_PVG_00/IO_VOUT_00`) and `IO_PIN_H2` (`IO_PVG_02/IO_VOUT_02`) must either remain unconfigured or also used as voltage outputs. Initializing it as any other output type will result in a `IO_E_GROUP_CONFLICT` error.

- The outputs will only be activated after enabling them via

```
IO_POWER_Set (IO_INT_PVG_VOUT_0_ENABLE,  
              IO_POWER_ON) ;  
IO_POWER_Set (IO_INT_PVG_VOUT_1_ENABLE,  
              IO_POWER_ON) ;
```

After activating the outputs, it is not possible to initialize any further PVG/VOut channels.

- Initialization/usage order:
 1. Initialize ALL needed voltage output channels.

2. Call `IO_POWER_Set(IO_INT_PVG_VOUT_x_ENABLE, IO_POWER_ON);`

3. Set desired output value with `IO_VOut_SetVoltage`.

- After enabling the outputs by calling `IO_POWER_Set`, all configured VOut channels will output the desired value, while all other pins within this group will output 0V constantly.
- When configuring a voltage output, the associated voltage feedback channel will also be configured.
- On initialization of a voltage output, a configuration for the PID controller can be specified via the parameter `pid_config`. If this parameter is `NULL`, following default values will be used:

```
IO_PID_CONFIG pid_config = {  
    .Kff = 1000,  
    .Kp = 200,  
    .Ki = 2500,  
    .Kd = 300,  
    .max_limit = 65535,  
    .min_limit = 0  
};
```

The PID configuration heavily depends on the load attached to the output and most likely need to be modified.

- The cycle time of the PID controller used for voltage outputs is 3ms.

VOut code examples

Please refer to section **Basic structure of an application** for understanding where to place the initialization and task function calls.

VOut initialization examples:

```
// Setup a voltage output with the default PID-
parameters
IO_VOut_Init( IO_VOUT_00
              , NULL           // use default PID-
parameters
              , NULL );       // use no PID
handler

// Enable all configured Voltage/PVG outputs
IO_POWER_Set( IO_INT_PVG_VOUT_0_ENABLE,
              IO_POWER_ON );
```

VOut task examples:

```
ubyte2 voltage_fb;
IO_VOut_SetVoltage( IO_VOUT_00
                   , 6200           // set
output voltage to 6200mV
                   , &voltage_fb); // variable
to store voltage feedback
```

Function Documentation

IO_ErrorType IO_VOut_DeInit (IO_PIN vout_channel)

Deinitializes one voltage output channel.

Parameters

vout_channel voltage output channel (`IO_VOUT_00 .. IO_VOUT_05`)

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_INVALID_CHANNEL_ID

the given channel id does not exist

IO_E_CHANNEL_NOT_CONFIGURED the given channel is not configured

Remarks

- The following channels form groups. A group always has to be configured as a whole.
 - `IO_VOUT_00 .. IO_VOUT_02`
 - `IO_VOUT_03 .. IO_VOUT_05`

IO_ErrorType

IO_VOut_Init (IO_PIN vout_channel, const IO_PID_CONFIG *const pid_config, ubyte1 *const pid_handle)

)

Setup one voltage output channel.

Parameters

- vout_channel** voltage output channel (`IO_VOUT_00` .. `IO_VOUT_05`)
- pid_config** Pointer to the data structure containing a PID configuration (optional). If NULL, default values will be used.
- pid_handle** PID handle of the used PID controller (optional).
Can be used to call PID functions. If NULL, no handle will be returned.

Returns

`IO_ErrorType`

Return values

- | | |
|--|---|
| <code>IO_E_OK</code> | everything fine |
| <code>IO_E_GROUP_CONFLICT</code> | configuring the output not allowed due to conflicts with other in/outputs in the same group |
| <code>IO_E_PID_NO_FREE_HANDLES</code> | No PID controller could be initialized |
| <code>IO_E_INVALID_CHANNEL_ID</code> | the channel id does not exist |
| <code>IO_E_CHANNEL_BUSY</code> | the ADC input channel is currently used by another function |
| <code>IO_E_DRIVER_NOT_INITIALIZED</code> | The common driver init function has not been |

IO_E_TASK_NO_FREE_SLOTS

called before

IO_E_SW_INTERNAL

No more free slots to setup task function

IO_E_CH_CAPABILITY

Internal software error

The ADC capability of this channel has not been activated

Remarks

The associated voltage feedback channel will also be configured for diagnosis.

The associated PID controller will be initialized and activated.

When NULL is passed as `pid_config`, following default configuration will be used:

```
IO_PID_CONFIG pid_config = {  
    .Kff = 1000,  
    .Kp = 200,  
    .Ki = 2500,  
    .Kd = 300,  
    .max_limit = 65535,  
    .min_limit = 0  
};
```

The cycle time of the PID controller is 3ms and not configurable.

The values `min_limit` and `max_limit` of the parameter `pid_config` are ignored by the PID controller for the voltage outputs.

- The following channels form groups. A group always has to be configured as a whole.
 - `IO_VOUT_00 .. IO_VOUT_02`
 - `IO_VOUT_03 .. IO_VOUT_05`

- Check the alternate functions of the pins used in each group. A pin can only be configured for one function at a time and it has to be the same function within the group - however mixing DO functionality together with VOUT functionality in the same group is a valid configuration and vice versa. The alternate functions can be found at [IO_Pins.h](#)

IO_ErrorType

```
IO_VOut_SetVoltage ( IO_PIN          vout_channel,
                    ubyte2          output_voltage,
                    ubyte2 *const voltage_fb
                    )
```

Sets the output voltage of one VOut channel.

Parameters

vout_channel voltage output channel ([IO_VOUT_00](#) .. [IO_VOUT_05](#))

output_voltage Output voltage in mV (0..32000)

voltage_fb Voltage feedback in mV (optional)

Returns

[IO_ErrorType](#)

Return values

IO_E_OK	everything fine
IO_E_INVALID_PARAMETER	parameter is out of range
IO_E_INVALID_CHANNEL_ID	the channel id does not exist
IO_E_CH_CAPABILITY	the given channel does not support

	the requested feature
IO_E_CHANNEL_NOT_CONFIGURED	the given channel is not configured
IO_E_CHANNEL_BUSY	the given channel is configured for another purpose
IO_E_VOUT_OUTPUT_DISABLED	the voltage output is disabled
IO_E_VOUT_SHORT_BATTERY	short to UBAT has been detected
IO_E_VOUT_SHORT_CIRCUIT	short to GND has been detected
IO_E_VOUT_PRECISION	the desired output voltage was not reached within the settling time
IO_E_PROT_ACTIVE	Protection is active and output value was automatically set to a value to protect the outputs
IO_E_PROT_REENABLE	The IO-Driver signals that the protection was disabled and the output functionality was restored
IO_E_ADC_INVALID	the received ADC value is corrupted

Remarks

The output voltage is dependent of the supply voltage. It is not possible to set an output voltage higher than the supply voltage.

An output voltage level close to the supply voltage can cause an **IO_E_VOUT_SHORT_BATTERY** error.

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

IO_WD.h File Reference

Functions

IO-Driver for the Window Watchdog. [More...](#)

```
#include "IO_Driver.h"
```

Functions

IO_ErrorType	IO_WD_Service (void)
Services the Window Watchdog.	

Detailed Description

IO-Driver for the Window Watchdog.

Contains a function to service the external Watchdog.
The Window Watchdog needs no initialization, as it is initialized automatically by **IO_Driver_Init**.

Function Documentation

IO_ErrorType IO_WD_Service (void)

Services the Window Watchdog.

This function services the CPU-external Window Watchdog. If the flag **IO_DRIVER_MODE_SERVICE_WD** was not set on IO-Driver configuration (see **IO_Driver.h**), this function must be called at least every 40ms, otherwise the Watchdog will deactivate the low-side digital outputs.

Remarks

- The Window Watchdog also has a lower time limit of 2ms, i.e. it must not be serviced with a period shorter than 2ms. This function handles the lower boundary automatically, that is, if it's called more frequent than 2ms it does only service the Watchdog if 2ms have passed.
- Even if **IO_DRIVER_MODE_SERVICE_WD** was set as mode on IO-Driver initialization, this function can be called. This can be useful if some operations (e.g. during initialization phase) take more time than 40ms, but the automatic servicing feature shall be used.

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_NOT_CONFIGURED

module has not
been initialized

IO_E_WD_TRIGGER_DISABLED

trigger has been
disabled

IO_E_SW_INTERNAL

permanently
a internal software
error occurred

Main Page	Related Pages	Data Structures	Files
File List	Globals		
inc			

Functions

IO_WDTimer.h File Reference

IO Driver functions for the CPU's Watchdog timer. [More...](#)

```
#include "IO_Driver.h"
```

Functions

IO_ErrorType **IO_WDTimer_Init** (**ubyte4** timeout)
Initialization of the Watchdog Timer.

IO_ErrorType **IO_WDTimer_DeInit** (void)
Disable the Watchdog Timer.

IO_ErrorType **IO_WDTimer_Service** (void)
Service the Watchdog timer.

Detailed Description

IO Driver functions for the CPU's Watchdog timer.

The Watchdog Timer (WDT) is a secure mechanism to overcome life- and dead-locks. An enabled WDT generates a reset for the system if not serviced in a configured time frame.

Remark

After the first watchdog timer overrun (ie. when the watchdog timer is not serviced within the configured timeout), the CPU will be reset and the application starts again. After the second overrun the CPU will be held in a reset state to avoid endless resets.

Note

These IO Driver functions are only available if **IO_Driver_Init()** is called without safety configuration i.e.:
`IO_Driver_Init(IO_DRIVER_MODE_DEFAULT, NULL);`

Watchdog timer code examples

Example for watchdog timer initialization:

```
IO_WDTimer_Init(350000); //setup the wdtimer  
    with a timeout of 350ms
```

Example for watchdog timer service:

```
IO_WDTimer_Service(); //service the watchdog  
    timer
```

Function Documentation

IO_ErrorType IO_WDTimer_DeInit (void)

Disable the Watchdog Timer.

disables the watchdog timer functionality.

Returns

IO_ErrorType

Return values

IO_E_OK

everything fine

IO_E_CHANNEL_NOT_CONFIGURED

internal Watchdog
not initialized

IO_E_WD_INT_ONLY_NON_SAFETY

IO_Driver is
safety relevant
configured,
therefore it's not
allowed to
disabled the
internal Watchdog

IO_ErrorType IO_WDTimer_Init (ubyte4 timeout)

Initialization of the Watchdog Timer.

The function

- configures the timeout of the watchdog timer
- enables the timer

Parameters

timeout timeout for the watchdog timer in us the CPU will be reseted if the watchdog timer is not serviced within this period

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_BUSY	Module already initialized
IO_E_INVALID_PARAMETER	Given parameter is invalid
IO_E_WD_INT_ONLY_NON_SAFETY	Watchdog already initialized via IO_Driver_Init()

IO_ErrorType IO_WDTimer_Service (void)

Service the Watchdog timer.

Service routine for the watchdog timer. If the time between two Service function calls exceeds the timeout (set with IO_WDTimer_Init) the CPU will be resetted.

Returns

IO_ErrorType

Return values

IO_E_OK	everything fine
IO_E_WD_INT_ONLY_NON_SAFETY	Watchdog is serviced automatically

Generated on Mon Nov 16 2020 16:59:47 for HY-TTC 30 Family C API Manual by

doxygen 1.8.2

[Main Page](#)[Related Pages](#)[Data Structures](#)[Files](#)[File List](#)[Globals](#)[inc](#)[Macros](#) | [Typedefs](#)

ptypes_xe167.h File Reference

Primitive data types. [More...](#)

Macros

```
#define FALSE ((bool)0)
```

```
#define TRUE (!FALSE)
```

```
#define NULL (0)
```

Typedefs

```
typedef unsigned char ubyte1
```

```
typedef unsigned int ubyte2
```

```
typedef unsigned long ubyte4
```

```
typedef unsigned long long ubyte8
```

typedef signed char **sbyte1**

typedef signed int **sbyte2**

typedef signed long **sbyte4**

typedef signed long long **sbyte8**

typedef float **float4**

typedef unsigned char **bool**

Detailed Description

Primitive data types.

This file defines the primitive data types used for the IO Driver

Macro Definition Documentation

#define FALSE ((bool)0)

FALSE value for boolean type

#define NULL (0)

NULL value, e.g. for invalid pointers

#define TRUE (!FALSE)

TRUE value for boolean type

Typedef Documentation

typedef unsigned char `bool`

boolean type, should only be set to 0 (FALSE) or 1 (TRUE)

typedef float `float4`

floating point, four bytes (32bit)

typedef signed char `sbyte1`

signed, length: one byte (8bit)

typedef signed int `sbyte2`

signed, length: two bytes (16bit)

typedef signed long `sbyte4`

signed, length: four bytes (32bit)

typedef signed long long `sbyte8`

signed, length: eight bytes (64bit)

typedef unsigned char [ubyte1](#)

unsigned, length: one byte (8bit)

typedef unsigned int [ubyte2](#)

unsigned, length: two bytes (16bit)

typedef unsigned long [ubyte4](#)

unsigned, length: four bytes (32bit)

typedef unsigned long long [ubyte8](#)

unsigned, length: eight bytes (64bit)



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page	Related Pages	Data Structures	Files
File List	Globals		
External	LogisticTypes	Types	

[Data Structures](#) | [Macros](#) | [Typedefs](#)

TypesGen.h File Reference

Types header file. [More...](#)

Data Structures

struct **can_id**
CAN ID structure. [More...](#)

Macros

#define **UINT32_ALL_BITS_SET** (0xFFFFFFFFllu)
Compiler instruction to create full-sized pointers.

Typedefs

typedef struct **can_id** **CanIdType**
CAN ID structure.

Detailed Description

Types header file.

Copyright (c) TTControl. All rights reserved. Confidential and proprietary.

Macro Definition Documentation

#define UINT32_ALL_BITS_SET (0xFFFFFFFFlu)

Compiler instruction to create full-sized pointers.

On some CPUs it is possible to select a memory model that creates pointers with a reduced size (e.g. CPUs where the bit width of the ALU is smaller than the bit-width of the address bus). For these memory models the below definition is important to create full-sized pointers to be able to address memory objects that are greater than the range of the reduced size pointer.

For environments where those memory models that operate with reduced size pointers do not exist or are not used the definition below can be empty



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page		Related Pages			Data Structures			Files				
File List		Globals										
All	Functions		Typedefs		Enumerations		Enumerator					
Macros												
_	a	b	c	d	f	i	n	s	t	u		

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- _ -

- `_diag_errortype` : [DIAG_Constants.h](#)
- `_io_driver_reset_reason` : [IO_Driver.h](#)

HY-TTC 30 Family C

API Manual D-TTC-X-G-

20-001

Main Page	Related Pages	Data Structures	Files
File List	Globals		
All	Functions	Typedefs	Enumerations
Macros			Enumerator
-	a	b	c
	d	f	i
	n	s	t
	u		

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- a -

- APDB_FLAGS_ABRD_ENABLE : [Apdb.h](#)
- APDB_FLAGS_CRC64_ENABLE : [Apdb.h](#)
- APDB_FLAGS_MULTI_APP : [Apdb.h](#)
- APDB_SIZE : [Apdb.h](#)
- APDB_VERSION : [Apdb.h](#)



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page		Related Pages		Data Structures		Files						
File List		Globals										
All	Functions	Typedefs	Enumerations		Enumerator							
Macros												
_	a	b	c	d	f	i	n	s	t	u		

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- b -

- bool : [ptypes_xe167.h](#)



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page		Related Pages		Data Structures		Files	
File List		Globals					
All	Functions	Typedefs	Enumerations		Enumerator		
Macros							
_	a	b	c	d	f	i	n s t u

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- C -

- CanIdType : [TypesGen.h](#)



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page		Related Pages		Data Structures		Files				
File List		Globals								
All	Functions	Typedefs	Enumerations		Enumerator					
Macros										
_	a	b	c	d	f	i	n	s	t	u

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- d -

- DIAG_E_ADC_5V2_SUPPLY : [DIAG_Constants.h](#)
- DIAG_E_ADC_KL30_CPU : [DIAG_Constants.h](#)
- DIAG_E_ADC_KL30_MAIN : [DIAG_Constants.h](#)
- DIAG_E_ADC_LIMITS : [DIAG_Constants.h](#)
- DIAG_E_ADC_SENSOR_SUPPLY : [DIAG_Constants.h](#)
- DIAG_E_APPL_SAFE_STATE : [DIAG_Constants.h](#)
- DIAG_E_CYCLE_TIME : [DIAG_Constants.h](#)
- DIAG_E_EXT_WD : [DIAG_Constants.h](#)
- DIAG_E_FREQ_STARTUP : [DIAG_Constants.h](#)
- DIAG_E_INIT_ERROR : [DIAG_Constants.h](#)
- DIAG_E_INT_WATCHDOG : [DIAG_Constants.h](#)
- DIAG_E_INVALID_DIAG_STATE : [DIAG_Constants.h](#)
- DIAG_E_INVALID_MAIN_STATE : [DIAG_Constants.h](#)
- DIAG_E_INVALID_STARTUP_STATE : [DIAG_Constants.h](#)
- DIAG_E_LS_PROT : [DIAG_Constants.h](#)
- DIAG_E_MEM_CarryFlag : [DIAG_Constants.h](#)
- DIAG_E_MEM_CLASS_B_TRAP : [DIAG_Constants.h](#)

- DIAG_E_MEM_DPRAM : [DIAG_Constants.h](#)
- DIAG_E_MEM_DSRAM : [DIAG_Constants.h](#)
- DIAG_E_MEM_NegativeFlag : [DIAG_Constants.h](#)
- DIAG_E_MEM_OverflowFlag : [DIAG_Constants.h](#)
- DIAG_E_MEM_PSRAM : [DIAG_Constants.h](#)
- DIAG_E_MEM_REGISTER : [DIAG_Constants.h](#)
- DIAG_E_MEM_SOFTBREAK_TRAP : [DIAG_Constants.h](#)
- DIAG_E_MEM_SR0_TRAP : [DIAG_Constants.h](#)
- DIAG_E_MEM_SYS_STACK_OF : [DIAG_Constants.h](#)
- DIAG_E_MEM_SYS_STACK_UF : [DIAG_Constants.h](#)
- DIAG_E_MEM_USER_STACK : [DIAG_Constants.h](#)
- DIAG_E_MEM_ZeroFlag : [DIAG_Constants.h](#)
- DIAG_E_NOERROR : [DIAG_Constants.h](#)
- DIAG_E_OVD : [DIAG_Constants.h](#)
- DIAG_E_OVD_STARTUP : [DIAG_Constants.h](#)
- DIAG_E_OVER_TEMPERATURE : [DIAG_Constants.h](#)
- DIAG_E_PLL_VCO_NOT_LOCKED : [DIAG_Constants.h](#)
- DIAG_E_PWD_LIMITS_FREQ : [DIAG_Constants.h](#)
- DIAG_E_PWD_LIMITS_PULSE_WIDTH : [DIAG_Constants.h](#)
- DIAG_E_PWM_CURRENT : [DIAG_Constants.h](#)
- DIAG_E_PWM_CURRENT_DEAD_TIME : [DIAG_Constants.h](#)
- DIAG_E_PWM_CURRENT_OFFS_DRIFT : [DIAG_Constants.h](#)
- DIAG_E_PWM_CURRENT_OFFSET : [DIAG_Constants.h](#)
- DIAG_E_PWM_CURRENT_ZERO : [DIAG_Constants.h](#)
- DIAG_E_PWM_LIMITS_RANGE : [DIAG_Constants.h](#)
- DIAG_E_PWM_LIMITS_TOL : [DIAG_Constants.h](#)
- DIAG_E_PWM_PERIOD_MISMATCH : [DIAG_Constants.h](#)
- DIAG_E_RPP : [DIAG_Constants.h](#)
- DIAG_E_SAFETY_SW_EXT : [DIAG_Constants.h](#)
- DIAG_E_SAFETY_SW_INT : [DIAG_Constants.h](#)
- DIAG_E_SAFETY_SW_SHUT_OFF : [DIAG_Constants.h](#)
- DIAG_E_SR_HighNibble : [DIAG_Constants.h](#)
- DIAG_E_SR_LowNibble : [DIAG_Constants.h](#)

- DIAG_E_SW_INTERNAL : **DIAG_Constants.h**
- DIAG_E_TIMEOUT : **DIAG_Constants.h**
- DIAG_E_WD_STARTUP : **DIAG_Constants.h**
- DIAG_EnableDischargeCircuit() : **DIAG_Functions.h**
- DIAG_EnterSafestate() : **DIAG_Functions.h**
- DIAG_ERR_CALLBACK : **DIAG_Constants.h**
- DIAG_ERR_NOACTION : **DIAG_Constants.h**
- DIAG_ERR_SAFESTATE : **DIAG_Constants.h**
- DIAG_ERRORCODE : **DIAG_Constants.h**
- DIAG_ErrorType : **DIAG_Constants.h**
- DIAG_STARTUP_TEST_ACTIVATE : **DIAG_Functions.h**
- DIAG_STARTUP_TEST_CTRL : **DIAG_Functions.h**
- DIAG_STARTUP_TEST_INHIBIT : **DIAG_Functions.h**
- DIAG_StartupTestCtrl() : **DIAG_Functions.h**
- DIAG_STATE_DISABLED : **DIAG_Constants.h**
- DIAG_STATE_INIT : **DIAG_Constants.h**
- DIAG_STATE_MAIN : **DIAG_Constants.h**
- DIAG_STATE_SAFE_STATE : **DIAG_Constants.h**
- DIAG_STATE_STARTUP : **DIAG_Constants.h**
- DIAG_Status() : **DIAG_Functions.h**



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page				Related Pages				Data Structures				Files													
File List				Globals																					
All		Functions			Typedefs			Enumerations			Enumerator														
Macros																									
_		a		b		c		d		f		i		n		s		t		u					

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- f -

- FALSE : [ptypes_xe167.h](#)
- float4 : [ptypes_xe167.h](#)



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page				Related Pages				Data Structures				Files											
File List				Globals																			
All		Functions				Typedefs				Enumerations				Enumerator									
Macros																							
_		a		b		c		d		f		i		n		s		t		u			

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- i -

- IO_ADC_00 : [IO_Pins.h](#)
- IO_ADC_01 : [IO_Pins.h](#)
- IO_ADC_10 : [IO_Pins.h](#)
- IO_ADC_11 : [IO_Pins.h](#)
- IO_ADC_12 : [IO_Pins.h](#)
- IO_ADC_13 : [IO_Pins.h](#)
- IO_ADC_14 : [IO_Pins.h](#)
- IO_ADC_15 : [IO_Pins.h](#)
- IO_ADC_20 : [IO_Pins.h](#)
- IO_ADC_21 : [IO_Pins.h](#)
- IO_ADC_22 : [IO_Pins.h](#)
- IO_ADC_23 : [IO_Pins.h](#)
- IO_ADC_24 : [IO_Pins.h](#)
- IO_ADC_25 : [IO_Pins.h](#)
- IO_ADC_26 : [IO_Pins.h](#)
- IO_ADC_27 : [IO_Pins.h](#)
- IO_ADC_28 : [IO_Pins.h](#)

- IO_ADC_29 : [IO_Pins.h](#)
- IO_ADC_30 : [IO_Pins.h](#)
- IO_ADC_31 : [IO_Pins.h](#)
- IO_ADC_32 : [IO_Pins.h](#)
- IO_ADC_33 : [IO_Pins.h](#)
- IO_ADC_34 : [IO_Pins.h](#)
- IO_ADC_35 : [IO_Pins.h](#)
- IO_ADC_36 : [IO_Pins.h](#)
- IO_ADC_37 : [IO_Pins.h](#)
- IO_ADC_38 : [IO_Pins.h](#)
- IO_ADC_39 : [IO_Pins.h](#)
- IO_ADC_40 : [IO_Pins.h](#)
- IO_ADC_41 : [IO_Pins.h](#)
- IO_ADC_5V2 : [IO_Pins.h](#)
- IO_ADC_ABSOLUTE : [IO_ADC.h](#)
- IO_ADC_BOARD_TEMP : [IO_Pins.h](#)
- IO_ADC_BoardTempFloat() : [IO_ADC.h](#)
- IO_ADC_BoardTempSbyte() : [IO_ADC.h](#)
- IO_ADC_ChannelDeInit() : [IO_ADC.h](#)
- IO_ADC_ChannelInit() : [IO_ADC.h](#)
- IO_ADC_CURRENT : [IO_ADC.h](#)
- IO_ADC_Get() : [IO_ADC.h](#)
- IO_ADC_NODE_ID_0 : [IO_Pins.h](#)
- IO_ADC_NODE_ID_1 : [IO_Pins.h](#)
- IO_ADC_RANGE_10V : [IO_ADC.h](#)
- IO_ADC_RANGE_5V : [IO_ADC.h](#)
- IO_ADC_RATIOMETRIC : [IO_ADC.h](#)
- IO_ADC_RESISTIVE : [IO_ADC.h](#)
- IO_ADC_SAFETY_CONF : [IO_ADC.h](#)
- IO_ADC_SENSOR_SUPPLY : [IO_Pins.h](#)
- IO_ADC_UBAT : [IO_Pins.h](#)
- IO_ADC_UBAT_CPU : [IO_Pins.h](#)
- IO_BRBL_CAN_ID : [IO_BRBL.h](#)
- IO_BRBL_CAN_PARAM : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_0 : [IO_BRBL.h](#)

- IO_BRBL_CUSTOM_DID_IDX_1 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_10 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_11 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_12 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_13 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_14 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_15 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_2 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_3 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_4 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_5 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_6 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_7 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_8 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_9 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_TBL_LEN : [IO_BRBL.h](#)
- IO_BRBL_GetCanParam() : [IO_BRBL.h](#)
- IO_BRBL_GetDid() : [IO_BRBL.h](#)
- IO_BRBL_GetXteaKey() : [IO_BRBL.h](#)
- IO_BRBL_Validate() : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_0 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_1 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_10 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_11 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_2 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_3 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_4 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_5 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_6 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_7 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_8 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_9 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_LEN : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_TBL_LEN : [IO_BRBL.h](#)
- IO_CAN_BAUDRATE_1000K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_100K : [IO_CAN.h](#)

- IO_CAN_BAUDRATE_10K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_125K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_20K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_250K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_25K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_500K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_50K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_800K : [IO_CAN.h](#)
- IO_CAN_CHANNEL_0 : [IO_Pins.h](#)
- IO_CAN_CHANNEL_1 : [IO_Pins.h](#)
- IO_CAN_ConfigFIFO() : [IO_CAN.h](#)
- IO_CAN_ConfigMsg() : [IO_CAN.h](#)
- IO_CAN_DATA_FRAME : [IO_CAN.h](#)
- IO_CAN_DeInit() : [IO_CAN.h](#)
- IO_CAN_DeInitHandle() : [IO_CAN.h](#)
- IO_CAN_EXT_FRAME : [IO_CAN.h](#)
- IO_CAN_FIFOStatus() : [IO_CAN.h](#)
- IO_CAN_Init() : [IO_CAN.h](#)
- IO_CAN_InitTimings() : [IO_CAN.h](#)
- IO_CAN_MSG_READ : [IO_CAN.h](#)
- IO_CAN_MSG_WRITE : [IO_CAN.h](#)
- IO_CAN_MsgStatus() : [IO_CAN.h](#)
- IO_CAN_ReadFIFO() : [IO_CAN.h](#)
- IO_CAN_ReadMsg() : [IO_CAN.h](#)
- IO_CAN_Status() : [IO_CAN.h](#)
- IO_CAN_STD_FRAME : [IO_CAN.h](#)
- IO_CAN_WriteFIFO() : [IO_CAN.h](#)
- IO_CAN_WriteMsg() : [IO_CAN.h](#)
- IO_Crypt_GetPseudoRandomNumber() : [IO_Crypt.h](#)
- IO_CRYPT_XTEA_KEY_LEN : [IO_Crypt.h](#)
- IO_Crypt_XteaDecipher() : [IO_Crypt.h](#)
- IO_Crypt_XteaDecipher32() : [IO_Crypt.h](#)
- IO_Crypt_XteaEncipher() : [IO_Crypt.h](#)
- IO_Crypt_XteaEncipher32() : [IO_Crypt.h](#)
- IO_DI_00 : [IO_Pins.h](#)
- IO_DI_01 : [IO_Pins.h](#)

- IO_DI_02 : [IO_Pins.h](#)
- IO_DI_03 : [IO_Pins.h](#)
- IO_DI_04 : [IO_Pins.h](#)
- IO_DI_05 : [IO_Pins.h](#)
- IO_DI_06 : [IO_Pins.h](#)
- IO_DI_07 : [IO_Pins.h](#)
- IO_DI_10 : [IO_Pins.h](#)
- IO_DI_11 : [IO_Pins.h](#)
- IO_DI_12 : [IO_Pins.h](#)
- IO_DI_13 : [IO_Pins.h](#)
- IO_DI_14 : [IO_Pins.h](#)
- IO_DI_15 : [IO_Pins.h](#)
- IO_DI_16 : [IO_Pins.h](#)
- IO_DI_17 : [IO_Pins.h](#)
- IO_DI_18 : [IO_Pins.h](#)
- IO_DI_19 : [IO_Pins.h](#)
- IO_DI_20 : [IO_Pins.h](#)
- IO_DI_21 : [IO_Pins.h](#)
- IO_DI_22 : [IO_Pins.h](#)
- IO_DI_23 : [IO_Pins.h](#)
- IO_DI_24 : [IO_Pins.h](#)
- IO_DI_25 : [IO_Pins.h](#)
- IO_DI_26 : [IO_Pins.h](#)
- IO_DI_27 : [IO_Pins.h](#)
- IO_DI_28 : [IO_Pins.h](#)
- IO_DI_29 : [IO_Pins.h](#)
- IO_DI_30 : [IO_Pins.h](#)
- IO_DI_31 : [IO_Pins.h](#)
- IO_DI_DeInit() : [IO_DIO.h](#)
- IO_DI_Get() : [IO_DIO.h](#)
- IO_DI_Init() : [IO_DIO.h](#)
- IO_DI_PD : [IO_DIO.h](#)
- IO_DI_PU : [IO_DIO.h](#)
- IO_DO_00 : [IO_Pins.h](#)
- IO_DO_01 : [IO_Pins.h](#)
- IO_DO_02 : [IO_Pins.h](#)

- IO_DO_03 : [IO_Pins.h](#)
- IO_DO_04 : [IO_Pins.h](#)
- IO_DO_05 : [IO_Pins.h](#)
- IO_DO_06 : [IO_Pins.h](#)
- IO_DO_07 : [IO_Pins.h](#)
- IO_DO_10 : [IO_Pins.h](#)
- IO_DO_11 : [IO_Pins.h](#)
- IO_DO_20 : [IO_Pins.h](#)
- IO_DO_21 : [IO_Pins.h](#)
- IO_DO_22 : [IO_Pins.h](#)
- IO_DO_23 : [IO_Pins.h](#)
- IO_DO_24 : [IO_Pins.h](#)
- IO_DO_25 : [IO_Pins.h](#)
- IO_DO_30 : [IO_Pins.h](#)
- IO_DO_31 : [IO_Pins.h](#)
- IO_DO_32 : [IO_Pins.h](#)
- IO_DO_33 : [IO_Pins.h](#)
- IO_DO_34 : [IO_Pins.h](#)
- IO_DO_35 : [IO_Pins.h](#)
- IO_DO_DeInit() : [IO_DIO.h](#)
- IO_DO_GetCur() : [IO_DIO.h](#)
- IO_DO_Init() : [IO_DIO.h](#)
- IO_DO_Set() : [IO_DIO.h](#)
- IO_DRIVER_DI_LIMITS : [IO_DIO.h](#)
- IO_Driver_GetAutoBaudrate() : [IO_Driver.h](#)
- IO_Driver_GetMode() : [IO_Driver.h](#)
- IO_Driver_GetResetStatus() : [IO_Driver.h](#)
- IO_Driver_GetResetStatus_ex() : [IO_Driver.h](#)
- IO_Driver_GetVersionOfBootloader() : [IO_Driver.h](#)
- IO_Driver_GetVersionOfDriver() : [IO_Driver.h](#)
- IO_Driver_Init() : [IO_Driver.h](#)
- IO_DRIVER_MODE_DEFAULT : [IO_Driver.h](#)
- IO_DRIVER_MODE_SERVICE_WD : [IO_Driver.h](#)
- IO_DRIVER_RESET_INFO : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON_PORST : [IO_Driver.h](#)

- IO_DRIVER_RESET_REASON_SW : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON_TRAP : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON_UNKNOWN : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON_WDT : [IO_Driver.h](#)
- IO_Driver_ResetToBootMode() : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_NA : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_PORST : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_SW : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_WD : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_WDT : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_APP : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_BL : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_NONE : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RSP_NONE : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RSP_SEND : [IO_Driver.h](#)
- IO_DRIVER_SAFETY_CONF : [IO_Driver.h](#)
- IO_DRIVER_SAFETY_SWITCH_EXT : [IO_Driver.h](#)
- IO_DRIVER_SAFETY_SWITCH_INT : [IO_Driver.h](#)
- IO_DRIVER_SAFETY_SWITCH_NONE : [IO_Driver.h](#)
- IO_DRIVER_SYSTEM_CLOCK : [IO_Driver.h](#)
- IO_Driver_TaskBegin() : [IO_Driver.h](#)
- IO_Driver_TaskEnd() : [IO_Driver.h](#)
- IO_DRIVER_TRAP_INFO : [IO_Driver.h](#)
- IO_E_ADC_CHANNEL_STARTUP : [IO_Constants.h](#)
- IO_E_ADC_INVALID : [IO_Constants.h](#)
- IO_E_BUSY : [IO_Constants.h](#)
- IO_E_CAN_BUS_OFF : [IO_Constants.h](#)
- IO_E_CAN_ERROR_PASSIVE : [IO_Constants.h](#)
- IO_E_CAN_FIFO_FULL : [IO_Constants.h](#)
- IO_E_CAN_INVALID_DATA : [IO_Constants.h](#)
- IO_E_CAN_MAX_HANDLES_REACHED : [IO_Constants.h](#)
- IO_E_CAN_MAX_MO_REACHED : [IO_Constants.h](#)
- IO_E_CAN_OLD_DATA : [IO_Constants.h](#)

- IO_E_CAN_OVERFLOW : [IO_Constants.h](#)
- IO_E_CAN_WRONG_HANDLE : [IO_Constants.h](#)
- IO_E_CH_CAPABILITY : [IO_Constants.h](#)
- IO_E_CHANNEL_BUSY : [IO_Constants.h](#)
- IO_E_CHANNEL_NOT_CONFIGURED : [IO_Constants.h](#)
- IO_E_DI_INVALID_LIMITS : [IO_Constants.h](#)
- IO_E_DI_INVALID_VOLTAGE : [IO_Constants.h](#)
- IO_E_DI_OPEN_LOAD : [IO_Constants.h](#)
- IO_E_DI_OPEN_LOAD_OR_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_DI_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_DI_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_DISCHARGE_FAILED : [IO_Constants.h](#)
- IO_E_DO_CHANNEL_STARTUP : [IO_Constants.h](#)
- IO_E_DO_CURRENT_INACCURATE : [IO_Constants.h](#)
- IO_E_DO_DIAG_TRANSIENT_OSC : [IO_Constants.h](#)
- IO_E_DO_OPEN_LOAD : [IO_Constants.h](#)
- IO_E_DO_OPEN_LOAD_OR_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_DO_OUTPUT_DISABLED : [IO_Constants.h](#)
- IO_E_DO_OUTPUT_STARTUP_ERROR : [IO_Constants.h](#)
- IO_E_DO_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_DO_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_DRIVER_INITIALIZED : [IO_Constants.h](#)
- IO_E_DRIVER_NOT_INITIALIZED : [IO_Constants.h](#)
- IO_E_DRV_SAFETY_CONF_NOT_CONFIG : [IO_Constants.h](#)
- IO_E_ECU_ALREADY_IN_SAFE_STATE : [IO_Constants.h](#)
- IO_E_EEPROM_BUFFER_FULL : [IO_Constants.h](#)
- IO_E_EEPROM_CRC_MISMATCH : [IO_Constants.h](#)
- IO_E_EEPROM_RANGE : [IO_Constants.h](#)
- IO_E_FET_PROTECTION : [IO_Constants.h](#)
- IO_E_GROUP_CONFLICT : [IO_Constants.h](#)
- IO_E_INVALID_CHANNEL_ID : [IO_Constants.h](#)
- IO_E_INVALID_CRC : [IO_Constants.h](#)
- IO_E_INVALID_DIAG_STATE : [IO_Constants.h](#)

- IO_E_INVALID_PARAMETER : [IO_Constants.h](#)
- IO_E_INVALID_SAFETY_CONFIG : [IO_Constants.h](#)
- IO_E_NO_SAFETY_SWITCH_CONFIGURED :
[IO_Constants.h](#)
- IO_E_NODEID_EEPROM_FALLBACK : [IO_Constants.h](#)
- IO_E_NODEID_EEPROM_INVALID : [IO_Constants.h](#)
- IO_E_NODEID_EEPROM_MISMATCH : [IO_Constants.h](#)
- IO_E_NODEID_PINS_INVALID : [IO_Constants.h](#)
- IO_E_NULL_POINTER : [IO_Constants.h](#)
- IO_E_OK : [IO_Constants.h](#)
- IO_E_PERIODIC_NOT_CONFIGURED : [IO_Constants.h](#)
- IO_E_PID_NO_FREE_HANDLES : [IO_Constants.h](#)
- IO_E_PID_USED : [IO_Constants.h](#)
- IO_E_PROT_ACTIVE : [IO_Constants.h](#)
- IO_E_PROT_FATAL : [IO_Constants.h](#)
- IO_E_PROT_PERMANENT_OFF : [IO_Constants.h](#)
- IO_E_PROT_REENABLE : [IO_Constants.h](#)
- IO_E_PROT_TEMP_OVERLOAD : [IO_Constants.h](#)
- IO_E_PROT_USER_OVERLOAD : [IO_Constants.h](#)
- IO_E_PVG_OUTPUT_DISABLED : [IO_Constants.h](#)
- IO_E_PVG_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_PVG_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_PWD_CAPTURE_ERROR : [IO_Constants.h](#)
- IO_E_PWD_HIGH_LEVEL : [IO_Constants.h](#)
- IO_E_PWD_LOW_LEVEL : [IO_Constants.h](#)
- IO_E_PWD_NOT_FINISHED : [IO_Constants.h](#)
- IO_E_PWD_TIMER_OVERFLOW : [IO_Constants.h](#)
- IO_E_PWM_CAPTURE_ERROR : [IO_Constants.h](#)
- IO_E_PWM_CHANNEL_STARTUP : [IO_Constants.h](#)
- IO_E_PWM_CURRENT_INACCURATE : [IO_Constants.h](#)
- IO_E_PWM_DIAG_TRANSIENT_OSC : [IO_Constants.h](#)
- IO_E_PWM_NOT_FINISHED : [IO_Constants.h](#)
- IO_E_PWM_OPEN_LOAD : [IO_Constants.h](#)
- IO_E_PWM_OPEN_LOAD_OR_SHORT_BATTERY :
[IO_Constants.h](#)
- IO_E_PWM_OUTPUT_DISABLED : [IO_Constants.h](#)

- IO_E_PWM_OUTPUT_HIGH : [IO_Constants.h](#)
- IO_E_PWM_OUTPUT_LOW : [IO_Constants.h](#)
- IO_E_PWM_OUTPUT_STARTUP_ERROR : [IO_Constants.h](#)
- IO_E_PWM_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_PWM_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_RESET_COUNTER_INVALID : [IO_Constants.h](#)
- IO_E_SAFETY_NOT_SUPPORTED : [IO_Constants.h](#)
- IO_E_SBRAM_CONTENT_INVALID : [IO_Constants.h](#)
- IO_E_SPI_BUFFER_FULL : [IO_Constants.h](#)
- IO_E_SPI_MAX_DEV_REACHED : [IO_Constants.h](#)
- IO_E_SW_INTERNAL : [IO_Constants.h](#)
- IO_E_SW_OUTPROT_SM : [IO_Constants.h](#)
- IO_E_TASK_NO_FREE_SLOTS : [IO_Constants.h](#)
- IO_E_UART_BUFFER_EMPTY : [IO_Constants.h](#)
- IO_E_UART_BUFFER_FULL : [IO_Constants.h](#)
- IO_E_UART_OVERFLOW : [IO_Constants.h](#)
- IO_E_UART_PARITY : [IO_Constants.h](#)
- IO_E_UNKNOWN : [IO_Constants.h](#)
- IO_E_VOUT_OUTPUT_DISABLED : [IO_Constants.h](#)
- IO_E_VOUT_PRECISION : [IO_Constants.h](#)
- IO_E_VOUT_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_VOUT_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_WD_INT_ONLY_NON_SAFETY : [IO_Constants.h](#)
- IO_E_WD_TRIGGER_DISABLED : [IO_Constants.h](#)
- IO_E_WD_TRIGGER_TEMPORARY_DISABLED : [IO_Constants.h](#)
- IO_E_WRONG_HW_TYPE : [IO_Constants.h](#)
- IO_EEPROM_DeInit() : [IO_EEPROM.h](#)
- IO_EEPROM_GetStatus() : [IO_EEPROM.h](#)
- IO_EEPROM_Init() : [IO_EEPROM.h](#)
- IO_EEPROM_PreloadDeInit() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadInit() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadRead() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadStatus() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadTask() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadWrite() : [IO_EEPROM_Preload.h](#)

- IO_EEPROM_Read() : [IO_EEPROM.h](#)
- IO_EEPROM_Write() : [IO_EEPROM.h](#)
- IO_ErrorType : [IO_Constants.h](#)
- IO_INT_DEV_CPU : [IO_Pins.h](#)
- IO_INT_PIN_5V2 : [IO_Pins.h](#)
- IO_INT_PIN_CAN_CH0 : [IO_Pins.h](#)
- IO_INT_PIN_CAN_CH1 : [IO_Pins.h](#)
- IO_INT_PIN_DRIVER : [IO_Pins.h](#)
- IO_INT_PIN_EEPROM : [IO_Pins.h](#)
- IO_INT_PIN_EXT_WD : [IO_Pins.h](#)
- IO_INT_PIN_PERIODIC : [IO_Pins.h](#)
- IO_INT_PIN_POWER : [IO_Pins.h](#)
- IO_INT_PIN_POWERSTAGE_ENABLE : [IO_Pins.h](#)
- IO_INT_PIN_PVG_VOUT_0_ENABLE : [IO_Pins.h](#)
- IO_INT_PIN_PVG_VOUT_1_ENABLE : [IO_Pins.h](#)
- IO_INT_PIN_RTC : [IO_Pins.h](#)
- IO_INT_PIN_SHIFT1_LB_HI : [IO_Pins.h](#)
- IO_INT_PIN_SHIFT1_LB_LO : [IO_Pins.h](#)
- IO_INT_PIN_SHIFT_LB_HI : [IO_Pins.h](#)
- IO_INT_PIN_SHIFT_LB_LO : [IO_Pins.h](#)
- IO_INT_PIN_TEMP : [IO_Pins.h](#)
- IO_INT_PIN_UART_CH0 : [IO_Pins.h](#)
- IO_INT_POWERSTAGE_ENABLE : [IO_Pins.h](#)
- IO_INT_PVG_VOUT_0_ENABLE : [IO_Pins.h](#)
- IO_INT_PVG_VOUT_1_ENABLE : [IO_Pins.h](#)
- IO_K15 : [IO_Pins.h](#)
- IO_LED_00 : [IO_Pins.h](#)
- IO_LED_01 : [IO_Pins.h](#)
- IO_LED_02 : [IO_Pins.h](#)
- IO_LED_03 : [IO_Pins.h](#)
- IO_LED_04 : [IO_Pins.h](#)
- IO_LED_05 : [IO_Pins.h](#)
- IO_LED_06 : [IO_Pins.h](#)
- IO_LED_07 : [IO_Pins.h](#)
- IO_LED_ChannelDelnit() : [IO_LED.h](#)
- IO_LED_Channellnit() : [IO_LED.h](#)

- IO_LED_Set() : [IO_LED.h](#)
- IO_NodeID_GetModifier() : [IO_NodeID.h](#)
- IO_NodeID_GetModifierStartup() : [IO_NodeID.h](#)
- IO_PID_CONFIG : [IO_PID.h](#)
- IO_PID_MAX_HANDLES : [IO_PID.h](#)
- IO_PID_SetIntegrator() : [IO_PID.h](#)
- IO_PIN : [IO_Pins.h](#)
- IO_PIN_A1 : [IO_Pins.h](#)
- IO_PIN_A3 : [IO_Pins.h](#)
- IO_PIN_A4 : [IO_Pins.h](#)
- IO_PIN_B1 : [IO_Pins.h](#)
- IO_PIN_B3 : [IO_Pins.h](#)
- IO_PIN_B4 : [IO_Pins.h](#)
- IO_PIN_C1 : [IO_Pins.h](#)
- IO_PIN_C3 : [IO_Pins.h](#)
- IO_PIN_C4 : [IO_Pins.h](#)
- IO_PIN_D1 : [IO_Pins.h](#)
- IO_PIN_D3 : [IO_Pins.h](#)
- IO_PIN_D4 : [IO_Pins.h](#)
- IO_PIN_E1 : [IO_Pins.h](#)
- IO_PIN_E2 : [IO_Pins.h](#)
- IO_PIN_E3 : [IO_Pins.h](#)
- IO_PIN_E4 : [IO_Pins.h](#)
- IO_PIN_F1 : [IO_Pins.h](#)
- IO_PIN_F2 : [IO_Pins.h](#)
- IO_PIN_F4 : [IO_Pins.h](#)
- IO_PIN_G1 : [IO_Pins.h](#)
- IO_PIN_G2 : [IO_Pins.h](#)
- IO_PIN_G4 : [IO_Pins.h](#)
- IO_PIN_H1 : [IO_Pins.h](#)
- IO_PIN_H2 : [IO_Pins.h](#)
- IO_PIN_H3 : [IO_Pins.h](#)
- IO_PIN_H4 : [IO_Pins.h](#)
- IO_PIN_J1 : [IO_Pins.h](#)
- IO_PIN_J2 : [IO_Pins.h](#)
- IO_PIN_J3 : [IO_Pins.h](#)

- IO_PIN_J4 : [IO_Pins.h](#)
- IO_PIN_K1 : [IO_Pins.h](#)
- IO_PIN_K2 : [IO_Pins.h](#)
- IO_PIN_K3 : [IO_Pins.h](#)
- IO_PIN_K4 : [IO_Pins.h](#)
- IO_PIN_L1 : [IO_Pins.h](#)
- IO_PIN_L2 : [IO_Pins.h](#)
- IO_POWER_DeInit() : [IO_POWER.h](#)
- IO_POWER_Get() : [IO_POWER.h](#)
- IO_POWER_Init() : [IO_POWER.h](#)
- IO_POWER_OFF : [IO_POWER.h](#)
- IO_POWER_ON : [IO_POWER.h](#)
- IO_POWER_Set() : [IO_POWER.h](#)
- IO_POWER_SetK15Threshold() : [IO_POWER.h](#)
- IO_PVG_00 : [IO_Pins.h](#)
- IO_PVG_01 : [IO_Pins.h](#)
- IO_PVG_02 : [IO_Pins.h](#)
- IO_PVG_03 : [IO_Pins.h](#)
- IO_PVG_04 : [IO_Pins.h](#)
- IO_PVG_05 : [IO_Pins.h](#)
- IO_PVG_DeInit() : [IO_PVG.h](#)
- IO_PVG_Init() : [IO_PVG.h](#)
- IO_PVG_SetOutput() : [IO_PVG.h](#)
- IO_PWD_00 : [IO_Pins.h](#)
- IO_PWD_01 : [IO_Pins.h](#)
- IO_PWD_02 : [IO_Pins.h](#)
- IO_PWD_03 : [IO_Pins.h](#)
- IO_PWD_10 : [IO_Pins.h](#)
- IO_PWD_11 : [IO_Pins.h](#)
- IO_PWD_12 : [IO_Pins.h](#)
- IO_PWD_13 : [IO_Pins.h](#)
- IO_PWD_20 : [IO_Pins.h](#)
- IO_PWD_21 : [IO_Pins.h](#)
- IO_PWD_22 : [IO_Pins.h](#)
- IO_PWD_23 : [IO_Pins.h](#)
- IO_PWD_BOTH_COUNT : [IO_PWD.h](#)

- IO_PWD_ComplexDelInit() : IO_PWD.h
- IO_PWD_ComplexGet() : IO_PWD.h
- IO_PWD_ComplexInit() : IO_PWD.h
- IO_PWD_CountDelInit() : IO_PWD.h
- IO_PWD_CountGet() : IO_PWD.h
- IO_PWD_CountInit() : IO_PWD.h
- IO_PWD_CountSet() : IO_PWD.h
- IO_PWD_CPLX_SAFETY_CONF : IO_PWD.h
- IO_PWD_DOWN_COUNT : IO_PWD.h
- IO_PWD_FALLING_COUNT : IO_PWD.h
- IO_PWD_FALLING_VAR : IO_PWD.h
- IO_PWD_FreqDelInit() : IO_PWD.h
- IO_PWD_FreqGet() : IO_PWD.h
- IO_PWD_FreqInit() : IO_PWD.h
- IO_PWD_HIGH_TIME : IO_PWD.h
- IO_PWD_INC_1_COUNT : IO_PWD.h
- IO_PWD_INC_2_COUNT : IO_PWD.h
- IO_PWD_INC_SAFETY_CONF : IO_PWD.h
- IO_PWD_IncDelInit() : IO_PWD.h
- IO_PWD_IncGet() : IO_PWD.h
- IO_PWD_IncInit() : IO_PWD.h
- IO_PWD_IncSet() : IO_PWD.h
- IO_PWD_LOW_TIME : IO_PWD.h
- IO_PWD_PD : IO_PWD.h
- IO_PWD_PERIOD_TIME : IO_PWD.h
- IO_PWD_PU : IO_PWD.h
- IO_PWD_PULSE_SAMPLES : IO_PWD.h
- IO_PWD_PulseDelInit() : IO_PWD.h
- IO_PWD_PulseFreqDelInit() : IO_PWD.h
- IO_PWD_PulseFreqGet() : IO_PWD.h
- IO_PWD_PulseFreqInit() : IO_PWD.h
- IO_PWD_PulseGet() : IO_PWD.h
- IO_PWD_PulseInit() : IO_PWD.h
- IO_PWD_RESOLUTION_0_2 : IO_PWD.h
- IO_PWD_RESOLUTION_0_4 : IO_PWD.h
- IO_PWD_RESOLUTION_0_8 : IO_PWD.h

- IO_PWD_RESOLUTION_1_6 : [IO_PWD.h](#)
- IO_PWD_RESOLUTION_3_2 : [IO_PWD.h](#)
- IO_PWD_RISING_COUNT : [IO_PWD.h](#)
- IO_PWD_RISING_VAR : [IO_PWD.h](#)
- IO_PWD_UP_COUNT : [IO_PWD.h](#)
- IO_PWM_00 : [IO_Pins.h](#)
- IO_PWM_01 : [IO_Pins.h](#)
- IO_PWM_02 : [IO_Pins.h](#)
- IO_PWM_03 : [IO_Pins.h](#)
- IO_PWM_04 : [IO_Pins.h](#)
- IO_PWM_05 : [IO_Pins.h](#)
- IO_PWM_10 : [IO_Pins.h](#)
- IO_PWM_11 : [IO_Pins.h](#)
- IO_PWM_CURRENT_QUEUE : [IO_PWM.h](#)
- IO_PWM_CURRENT_QUEUE_MAX : [IO_PWM.h](#)
- IO_PWM_CURRENT_SAFETY_CONF : [IO_PWM.h](#)
- IO_PWM_CurrentDeInit() : [IO_PWM.h](#)
- IO_PWM_CurrentInit() : [IO_PWM.h](#)
- IO_PWM_DeInit() : [IO_PWM.h](#)
- IO_PWM_GetCur() : [IO_PWM.h](#)
- IO_PWM_GetCurQueue() : [IO_PWM.h](#)
- IO_PWM_Init() : [IO_PWM.h](#)
- IO_PWM_SAFETY_CONF : [IO_PWM.h](#)
- IO_PWM_SetCur() : [IO_PWM.h](#)
- IO_PWM_SetDuty() : [IO_PWM.h](#)
- IO_RTC_GetTimeUS() : [IO_RTC.h](#)
- IO_RTC_Init() : [IO_RTC.h](#)
- IO_RTC_PeriodicDeInit() : [IO_RTC.h](#)
- IO_RTC_PeriodicInit() : [IO_RTC.h](#)
- IO_RTC_StartTime() : [IO_RTC.h](#)
- IO_SAFETY_SWITCH_0 : [IO_Pins.h](#)
- IO_SAFETY_SWITCH_1 : [IO_Pins.h](#)
- IO_SAFETY_SWITCH_NONE : [IO_Driver.h](#)
- IO_UART : [IO_Pins.h](#)
- IO_UART_DeInit() : [IO_UART.h](#)
- IO_UART_GetRxStatus() : [IO_UART.h](#)

- IO_UART_GetTxStatus() : [IO_UART.h](#)
- IO_UART_Init() : [IO_UART.h](#)
- IO_UART_Read() : [IO_UART.h](#)
- IO_UART_Task() : [IO_UART.h](#)
- IO_UART_Write() : [IO_UART.h](#)
- IO_VOUT_00 : [IO_Pins.h](#)
- IO_VOUT_01 : [IO_Pins.h](#)
- IO_VOUT_02 : [IO_Pins.h](#)
- IO_VOUT_03 : [IO_Pins.h](#)
- IO_VOUT_04 : [IO_Pins.h](#)
- IO_VOUT_05 : [IO_Pins.h](#)
- IO_VOut_DeInit() : [IO_Vout.h](#)
- IO_VOut_Init() : [IO_Vout.h](#)
- IO_VOut_SetVoltage() : [IO_Vout.h](#)
- IO_WD_Service() : [IO_WD.h](#)
- IO_WDTimer_DeInit() : [IO_WDTimer.h](#)
- IO_WDTimer_Init() : [IO_WDTimer.h](#)
- IO_WDTimer_Service() : [IO_WDTimer.h](#)



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page				Related Pages				Data Structures				Files		
File List				Globals										
All		Functions			Typedefs			Enumerations			Enumerator			
Macros														
-	a	b	c	d	f	i	n	s	t	u				

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- n -

- NULL : [ptypes_xe167.h](#)

HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page				Related Pages				Data Structures				Files			
File List				Globals											
All		Functions			Typedefs			Enumerations			Enumerator				
Macros															
-	a	b	c	d	f	i	n	s	t	u					

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- s -

- sbyte1 : [ptypes_xe167.h](#)
- sbyte2 : [ptypes_xe167.h](#)
- sbyte4 : [ptypes_xe167.h](#)
- sbyte8 : [ptypes_xe167.h](#)



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page				Related Pages				Data Structures				Files													
File List				Globals																					
All		Functions				Typedefs				Enumerations				Enumerator											
Macros																									
_		a		b		c		d		f		i		n		s		t		u					

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- t -

- TRUE : [ptypes_xe167.h](#)

HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page		Related Pages		Data Structures		Files						
File List		Globals										
All	Functions	Typedefs	Enumerations		Enumerator							
Macros												
_	a	b	c	d	f	i	n	s	t	u		

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- u -

- `ubyte1` : [ptypes_xe167.h](#)
- `ubyte2` : [ptypes_xe167.h](#)
- `ubyte4` : [ptypes_xe167.h](#)
- `ubyte8` : [ptypes_xe167.h](#)
- `UINT32_ALL_BITS_SET` : [TypesGen.h](#)

HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page		Related Pages		Data Structures		Files
File List		Globals				
All	Functions	Typedefs	Enumerations	Enumerator		
Macros						
d	i					

- d -

- DIAG_EnableDischargeCircuit() : [DIAG_Functions.h](#)
- DIAG_EnterSafestate() : [DIAG_Functions.h](#)
- DIAG_StartupTestCtrl() : [DIAG_Functions.h](#)
- DIAG_Status() : [DIAG_Functions.h](#)

- i -

- IO_ADC_BoardTempFloat() : [IO_ADC.h](#)
- IO_ADC_BoardTempSbyte() : [IO_ADC.h](#)
- IO_ADC_ChannelDeInit() : [IO_ADC.h](#)
- IO_ADC_ChannelInit() : [IO_ADC.h](#)
- IO_ADC_Get() : [IO_ADC.h](#)
- IO_BRBL_GetCanParam() : [IO_BRBL.h](#)
- IO_BRBL_GetDid() : [IO_BRBL.h](#)
- IO_BRBL_GetXteaKey() : [IO_BRBL.h](#)
- IO_BRBL_Validate() : [IO_BRBL.h](#)
- IO_CAN_ConfigFIFO() : [IO_CAN.h](#)
- IO_CAN_ConfigMsg() : [IO_CAN.h](#)

- IO_CAN_DeInit() : **IO_CAN.h**
- IO_CAN_DeInitHandle() : **IO_CAN.h**
- IO_CAN_FIFOStatus() : **IO_CAN.h**
- IO_CAN_Init() : **IO_CAN.h**
- IO_CAN_InitTimings() : **IO_CAN.h**
- IO_CAN_MsgStatus() : **IO_CAN.h**
- IO_CAN_ReadFIFO() : **IO_CAN.h**
- IO_CAN_ReadMsg() : **IO_CAN.h**
- IO_CAN_Status() : **IO_CAN.h**
- IO_CAN_WriteFIFO() : **IO_CAN.h**
- IO_CAN_WriteMsg() : **IO_CAN.h**
- IO_Crypt_GetPseudoRandomNumber() : **IO_Crypt.h**
- IO_Crypt_XteaDecipher() : **IO_Crypt.h**
- IO_Crypt_XteaDecipher32() : **IO_Crypt.h**
- IO_Crypt_XteaEncipher() : **IO_Crypt.h**
- IO_Crypt_XteaEncipher32() : **IO_Crypt.h**
- IO_DI_DeInit() : **IO_DIO.h**
- IO_DI_Get() : **IO_DIO.h**
- IO_DI_Init() : **IO_DIO.h**
- IO_DO_DeInit() : **IO_DIO.h**
- IO_DO_GetCur() : **IO_DIO.h**
- IO_DO_Init() : **IO_DIO.h**
- IO_DO_Set() : **IO_DIO.h**
- IO_Driver_GetAutoBaudrate() : **IO_Driver.h**
- IO_Driver_GetMode() : **IO_Driver.h**
- IO_Driver_GetResetStatus() : **IO_Driver.h**
- IO_Driver_GetResetStatus_ex() : **IO_Driver.h**
- IO_Driver_GetVersionOfBootloader() : **IO_Driver.h**
- IO_Driver_GetVersionOfDriver() : **IO_Driver.h**
- IO_Driver_Init() : **IO_Driver.h**
- IO_Driver_ResetToBootMode() : **IO_Driver.h**
- IO_Driver_TaskBegin() : **IO_Driver.h**
- IO_Driver_TaskEnd() : **IO_Driver.h**
- IO_EEPROM_DeInit() : **IO_EEPROM.h**
- IO_EEPROM_GetStatus() : **IO_EEPROM.h**
- IO_EEPROM_Init() : **IO_EEPROM.h**

- IO_EEPROM_PreloadDelInit() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadInit() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadRead() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadStatus() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadTask() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_PreloadWrite() : [IO_EEPROM_Preload.h](#)
- IO_EEPROM_Read() : [IO_EEPROM.h](#)
- IO_EEPROM_Write() : [IO_EEPROM.h](#)
- IO_LED_ChannelDelInit() : [IO_LED.h](#)
- IO_LED_ChannelInit() : [IO_LED.h](#)
- IO_LED_Set() : [IO_LED.h](#)
- IO_NodeID_GetModifier() : [IO_NodeID.h](#)
- IO_NodeID_GetModifierStartup() : [IO_NodeID.h](#)
- IO_PID_SetIntegrator() : [IO_PID.h](#)
- IO_POWER_DelInit() : [IO_POWER.h](#)
- IO_POWER_Get() : [IO_POWER.h](#)
- IO_POWER_Init() : [IO_POWER.h](#)
- IO_POWER_Set() : [IO_POWER.h](#)
- IO_POWER_SetK15Threshold() : [IO_POWER.h](#)
- IO_PVG_DelInit() : [IO_PVG.h](#)
- IO_PVG_Init() : [IO_PVG.h](#)
- IO_PVG_SetOutput() : [IO_PVG.h](#)
- IO_PWD_ComplexDelInit() : [IO_PWD.h](#)
- IO_PWD_ComplexGet() : [IO_PWD.h](#)
- IO_PWD_ComplexInit() : [IO_PWD.h](#)
- IO_PWD_CountDelInit() : [IO_PWD.h](#)
- IO_PWD_CountGet() : [IO_PWD.h](#)
- IO_PWD_CountInit() : [IO_PWD.h](#)
- IO_PWD_CountSet() : [IO_PWD.h](#)
- IO_PWD_FreqDelInit() : [IO_PWD.h](#)
- IO_PWD_FreqGet() : [IO_PWD.h](#)
- IO_PWD_FreqInit() : [IO_PWD.h](#)
- IO_PWD_IncDelInit() : [IO_PWD.h](#)
- IO_PWD_IncGet() : [IO_PWD.h](#)
- IO_PWD_IncInit() : [IO_PWD.h](#)
- IO_PWD_IncSet() : [IO_PWD.h](#)

- IO_PWD_PulseDelInit() : **IO_PWD.h**
 - IO_PWD_PulseFreqDelInit() : **IO_PWD.h**
 - IO_PWD_PulseFreqGet() : **IO_PWD.h**
 - IO_PWD_PulseFreqInit() : **IO_PWD.h**
 - IO_PWD_PulseGet() : **IO_PWD.h**
 - IO_PWD_PulseInit() : **IO_PWD.h**
 - IO_PWM_CurrentDelInit() : **IO_PWM.h**
 - IO_PWM_CurrentInit() : **IO_PWM.h**
 - IO_PWM_DelInit() : **IO_PWM.h**
 - IO_PWM_GetCur() : **IO_PWM.h**
 - IO_PWM_GetCurQueue() : **IO_PWM.h**
 - IO_PWM_Init() : **IO_PWM.h**
 - IO_PWM_SetCur() : **IO_PWM.h**
 - IO_PWM_SetDuty() : **IO_PWM.h**
 - IO_RTC_GetTimeUS() : **IO_RTC.h**
 - IO_RTC_Init() : **IO_RTC.h**
 - IO_RTC_PeriodicDelInit() : **IO_RTC.h**
 - IO_RTC_PeriodicInit() : **IO_RTC.h**
 - IO_RTC_StartTime() : **IO_RTC.h**
 - IO_UART_DelInit() : **IO_UART.h**
 - IO_UART_GetRxStatus() : **IO_UART.h**
 - IO_UART_GetTxStatus() : **IO_UART.h**
 - IO_UART_Init() : **IO_UART.h**
 - IO_UART_Read() : **IO_UART.h**
 - IO_UART_Task() : **IO_UART.h**
 - IO_UART_Write() : **IO_UART.h**
 - IO_VOut_DelInit() : **IO_Vout.h**
 - IO_VOut_Init() : **IO_Vout.h**
 - IO_VOut_SetVoltage() : **IO_Vout.h**
 - IO_WD_Service() : **IO_WD.h**
 - IO_WDTimer_DelInit() : **IO_WDTimer.h**
 - IO_WDTimer_Init() : **IO_WDTimer.h**
 - IO_WDTimer_Service() : **IO_WDTimer.h**
-

HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page				Related Pages				Data Structures				Files											
File List				Globals																			
All		Functions				Typedefs				Enumerations				Enumerator									
Macros																							
b		c		d		f		i		s		u											

- b -

- bool : [ptypes_xe167.h](#)

- c -

- CanIdType : [TypesGen.h](#)

- d -

- DIAG_ERR_CALLBACK : [DIAG_Constants.h](#)
- DIAG_ERRORCODE : [DIAG_Constants.h](#)
- DIAG_ErrorType : [DIAG_Constants.h](#)

- f -

- float4 : [ptypes_xe167.h](#)

- i -

- IO_ADC_SAFETY_CONF : [IO_ADC.h](#)
- IO_BRBL_CAN_ID : [IO_BRBL.h](#)
- IO_BRBL_CAN_PARAM : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID : [IO_BRBL.h](#)
- IO_CAN_DATA_FRAME : [IO_CAN.h](#)
- IO_DRIVER_DI_LIMITS : [IO_DIO.h](#)
- IO_DRIVER_RESET_INFO : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON : [IO_Driver.h](#)
- IO_DRIVER_SAFETY_CONF : [IO_Driver.h](#)
- IO_DRIVER_TRAP_INFO : [IO_Driver.h](#)
- IO_ErrorType : [IO_Constants.h](#)
- IO_PID_CONFIG : [IO_PID.h](#)
- IO_PWD_CPLX_SAFETY_CONF : [IO_PWD.h](#)
- IO_PWD_INC_SAFETY_CONF : [IO_PWD.h](#)
- IO_PWD_PULSE_SAMPLES : [IO_PWD.h](#)
- IO_PWM_CURRENT_QUEUE : [IO_PWM.h](#)
- IO_PWM_CURRENT_SAFETY_CONF : [IO_PWM.h](#)
- IO_PWM_SAFETY_CONF : [IO_PWM.h](#)

- S -

- sbyte1 : [ptypes_xe167.h](#)
- sbyte2 : [ptypes_xe167.h](#)
- sbyte4 : [ptypes_xe167.h](#)
- sbyte8 : [ptypes_xe167.h](#)

- U -

- ubyte1 : [ptypes_xe167.h](#)
- ubyte2 : [ptypes_xe167.h](#)
- ubyte4 : [ptypes_xe167.h](#)
- ubyte8 : [ptypes_xe167.h](#)

HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page		Related Pages		Data Structures		Files
File List		Globals				
All	Functions	Typedefs	Enumerations	Enumerator		
Macros						

- `_diag_errortype` : [DIAG_Constants.h](#)
- `_io_driver_reset_reason` : [IO_Driver.h](#)
- `DIAG_STARTUP_TEST_CTRL` : [DIAG_Functions.h](#)
- `IO_PIN` : [IO_Pins.h](#)



HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page		Related Pages		Data Structures		Files
File List		Globals				
All	Functions	Typedefs	Enumerations	Enumerator		
Macros						
d	i					

- d -

- DIAG_E_ADC_5V2_SUPPLY : [DIAG_Constants.h](#)
- DIAG_E_ADC_KL30_CPU : [DIAG_Constants.h](#)
- DIAG_E_ADC_KL30_MAIN : [DIAG_Constants.h](#)
- DIAG_E_ADC_LIMITS : [DIAG_Constants.h](#)
- DIAG_E_ADC_SENSOR_SUPPLY : [DIAG_Constants.h](#)
- DIAG_E_APPL_SAFE_STATE : [DIAG_Constants.h](#)
- DIAG_E_CYCLE_TIME : [DIAG_Constants.h](#)
- DIAG_E_EXT_WD : [DIAG_Constants.h](#)
- DIAG_E_FREQ_STARTUP : [DIAG_Constants.h](#)
- DIAG_E_INIT_ERROR : [DIAG_Constants.h](#)
- DIAG_E_INT_WATCHDOG : [DIAG_Constants.h](#)
- DIAG_E_INVALID_DIAG_STATE : [DIAG_Constants.h](#)
- DIAG_E_INVALID_MAIN_STATE : [DIAG_Constants.h](#)
- DIAG_E_INVALID_STARTUP_STATE : [DIAG_Constants.h](#)
- DIAG_E_LS_PROT : [DIAG_Constants.h](#)
- DIAG_E_MEM_CarryFlag : [DIAG_Constants.h](#)
- DIAG_E_MEM_CLASS_B_TRAP : [DIAG_Constants.h](#)
- DIAG_E_MEM_DPRAM : [DIAG_Constants.h](#)

- DIAG_E_MEM_DSRAM : **DIAG_Constants.h**
- DIAG_E_MEM_NegativeFlag : **DIAG_Constants.h**
- DIAG_E_MEM_OverflowFlag : **DIAG_Constants.h**
- DIAG_E_MEM_PSRAM : **DIAG_Constants.h**
- DIAG_E_MEM_REGISTER : **DIAG_Constants.h**
- DIAG_E_MEM_SOFTBREAK_TRAP : **DIAG_Constants.h**
- DIAG_E_MEM_SR0_TRAP : **DIAG_Constants.h**
- DIAG_E_MEM_SYS_STACK_OF : **DIAG_Constants.h**
- DIAG_E_MEM_SYS_STACK_UF : **DIAG_Constants.h**
- DIAG_E_MEM_USER_STACK : **DIAG_Constants.h**
- DIAG_E_MEM_ZeroFlag : **DIAG_Constants.h**
- DIAG_E_NOERROR : **DIAG_Constants.h**
- DIAG_E_OVD : **DIAG_Constants.h**
- DIAG_E_OVD_STARTUP : **DIAG_Constants.h**
- DIAG_E_OVER_TEMPERATURE : **DIAG_Constants.h**
- DIAG_E_PLL_VCO_NOT_LOCKED : **DIAG_Constants.h**
- DIAG_E_PWD_LIMITS_FREQ : **DIAG_Constants.h**
- DIAG_E_PWD_LIMITS_PULSE_WIDTH : **DIAG_Constants.h**
- DIAG_E_PWM_CURRENT : **DIAG_Constants.h**
- DIAG_E_PWM_CURRENT_DEAD_TIME : **DIAG_Constants.h**
- DIAG_E_PWM_CURRENT_OFFSETS_DRIFT : **DIAG_Constants.h**
- DIAG_E_PWM_CURRENT_OFFSET : **DIAG_Constants.h**
- DIAG_E_PWM_CURRENT_ZERO : **DIAG_Constants.h**
- DIAG_E_PWM_LIMITS_RANGE : **DIAG_Constants.h**
- DIAG_E_PWM_LIMITS_TOL : **DIAG_Constants.h**
- DIAG_E_PWM_PERIOD_MISMATCH : **DIAG_Constants.h**
- DIAG_E_RPP : **DIAG_Constants.h**
- DIAG_E_SAFETY_SW_EXT : **DIAG_Constants.h**
- DIAG_E_SAFETY_SW_INT : **DIAG_Constants.h**
- DIAG_E_SAFETY_SW_SHUT_OFF : **DIAG_Constants.h**
- DIAG_E_SR_HighNibble : **DIAG_Constants.h**
- DIAG_E_SR_LowNibble : **DIAG_Constants.h**
- DIAG_E_SW_INTERNAL : **DIAG_Constants.h**

- DIAG_E_TIMEOUT : [DIAG_Constants.h](#)
- DIAG_E_WD_STARTUP : [DIAG_Constants.h](#)
- DIAG_STARTUP_TEST_ACTIVATE : [DIAG_Functions.h](#)
- DIAG_STARTUP_TEST_INHIBIT : [DIAG_Functions.h](#)

- i -

- IO_DRIVER_RESET_REASON_PORST : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON_SW : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON_TRAP : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON_UNKNOWN : [IO_Driver.h](#)
- IO_DRIVER_RESET_REASON_WDT : [IO_Driver.h](#)
- IO_INT_DEV_CPU : [IO_Pins.h](#)
- IO_INT_PIN_5V2 : [IO_Pins.h](#)
- IO_INT_PIN_CAN_CH0 : [IO_Pins.h](#)
- IO_INT_PIN_CAN_CH1 : [IO_Pins.h](#)
- IO_INT_PIN_DRIVER : [IO_Pins.h](#)
- IO_INT_PIN_EEPROM : [IO_Pins.h](#)
- IO_INT_PIN_EXT_WD : [IO_Pins.h](#)
- IO_INT_PIN_PERIODIC : [IO_Pins.h](#)
- IO_INT_PIN_POWER : [IO_Pins.h](#)
- IO_INT_PIN_POWERSTAGE_ENABLE : [IO_Pins.h](#)
- IO_INT_PIN_PVG_VOUT_0_ENABLE : [IO_Pins.h](#)
- IO_INT_PIN_PVG_VOUT_1_ENABLE : [IO_Pins.h](#)
- IO_INT_PIN_RTC : [IO_Pins.h](#)
- IO_INT_PIN_SHIFT1_LB_HI : [IO_Pins.h](#)
- IO_INT_PIN_SHIFT1_LB_LO : [IO_Pins.h](#)
- IO_INT_PIN_SHIFT_LB_HI : [IO_Pins.h](#)
- IO_INT_PIN_SHIFT_LB_LO : [IO_Pins.h](#)
- IO_INT_PIN_TEMP : [IO_Pins.h](#)
- IO_INT_PIN_UART_CH0 : [IO_Pins.h](#)
- IO_PIN_A1 : [IO_Pins.h](#)
- IO_PIN_A3 : [IO_Pins.h](#)
- IO_PIN_A4 : [IO_Pins.h](#)
- IO_PIN_B1 : [IO_Pins.h](#)
- IO_PIN_B3 : [IO_Pins.h](#)

- IO_PIN_B4 : [IO_Pins.h](#)
- IO_PIN_C1 : [IO_Pins.h](#)
- IO_PIN_C3 : [IO_Pins.h](#)
- IO_PIN_C4 : [IO_Pins.h](#)
- IO_PIN_D1 : [IO_Pins.h](#)
- IO_PIN_D3 : [IO_Pins.h](#)
- IO_PIN_D4 : [IO_Pins.h](#)
- IO_PIN_E1 : [IO_Pins.h](#)
- IO_PIN_E2 : [IO_Pins.h](#)
- IO_PIN_E3 : [IO_Pins.h](#)
- IO_PIN_E4 : [IO_Pins.h](#)
- IO_PIN_F1 : [IO_Pins.h](#)
- IO_PIN_F2 : [IO_Pins.h](#)
- IO_PIN_F4 : [IO_Pins.h](#)
- IO_PIN_G1 : [IO_Pins.h](#)
- IO_PIN_G2 : [IO_Pins.h](#)
- IO_PIN_G4 : [IO_Pins.h](#)
- IO_PIN_H1 : [IO_Pins.h](#)
- IO_PIN_H2 : [IO_Pins.h](#)
- IO_PIN_H3 : [IO_Pins.h](#)
- IO_PIN_H4 : [IO_Pins.h](#)
- IO_PIN_J1 : [IO_Pins.h](#)
- IO_PIN_J2 : [IO_Pins.h](#)
- IO_PIN_J3 : [IO_Pins.h](#)
- IO_PIN_J4 : [IO_Pins.h](#)
- IO_PIN_K1 : [IO_Pins.h](#)
- IO_PIN_K2 : [IO_Pins.h](#)
- IO_PIN_K3 : [IO_Pins.h](#)
- IO_PIN_K4 : [IO_Pins.h](#)
- IO_PIN_L1 : [IO_Pins.h](#)
- IO_PIN_L2 : [IO_Pins.h](#)

Main Page		Related Pages		Data Structures		Files	
File List		Globals					
All	Functions	Typedefs	Enumerations	Enumerator			
Macros							
a	d	f	i	n	t	u	

- a -

- APDB_FLAGS_ABRD_ENABLE : [Apdb.h](#)
- APDB_FLAGS_CRC64_ENABLE : [Apdb.h](#)
- APDB_FLAGS_MULTI_APP : [Apdb.h](#)
- APDB_SIZE : [Apdb.h](#)
- APDB_VERSION : [Apdb.h](#)

Main Page		Related Pages		Data Structures		Files	
File List		Globals					
All	Functions	Typedefs	Enumerations	Enumerator			
Macros							
a	d	f	i	n	t	u	

- d -

- DIAG_ERR_NOACTION : [DIAG_Constants.h](#)
- DIAG_ERR_SAFESTATE : [DIAG_Constants.h](#)
- DIAG_STATE_DISABLED : [DIAG_Constants.h](#)
- DIAG_STATE_INIT : [DIAG_Constants.h](#)
- DIAG_STATE_MAIN : [DIAG_Constants.h](#)
- DIAG_STATE_SAFE_STATE : [DIAG_Constants.h](#)
- DIAG_STATE_STARTUP : [DIAG_Constants.h](#)

HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page		Related Pages		Data Structures		Files	
File List		Globals					
All	Functions	Typedefs	Enumerations	Enumerator			
Macros							
a	d	f	i	n	t	u	

- f -

- FALSE : [ptypes_xe167.h](#)

HY-TTC 30 Family C

API Manual D-TTC-X-G-20-001

Main Page		Related Pages		Data Structures		Files	
File List		Globals					
All	Functions	Typedefs	Enumerations		Enumerator		
Macros							
a	d	f	i	n	t	u	

- i -

- IO_ADC_00 : [IO_Pins.h](#)
- IO_ADC_01 : [IO_Pins.h](#)
- IO_ADC_10 : [IO_Pins.h](#)
- IO_ADC_11 : [IO_Pins.h](#)
- IO_ADC_12 : [IO_Pins.h](#)
- IO_ADC_13 : [IO_Pins.h](#)
- IO_ADC_14 : [IO_Pins.h](#)
- IO_ADC_15 : [IO_Pins.h](#)
- IO_ADC_20 : [IO_Pins.h](#)
- IO_ADC_21 : [IO_Pins.h](#)
- IO_ADC_22 : [IO_Pins.h](#)
- IO_ADC_23 : [IO_Pins.h](#)
- IO_ADC_24 : [IO_Pins.h](#)
- IO_ADC_25 : [IO_Pins.h](#)
- IO_ADC_26 : [IO_Pins.h](#)
- IO_ADC_27 : [IO_Pins.h](#)
- IO_ADC_28 : [IO_Pins.h](#)
- IO_ADC_29 : [IO_Pins.h](#)

- IO_ADC_30 : [IO_Pins.h](#)
- IO_ADC_31 : [IO_Pins.h](#)
- IO_ADC_32 : [IO_Pins.h](#)
- IO_ADC_33 : [IO_Pins.h](#)
- IO_ADC_34 : [IO_Pins.h](#)
- IO_ADC_35 : [IO_Pins.h](#)
- IO_ADC_36 : [IO_Pins.h](#)
- IO_ADC_37 : [IO_Pins.h](#)
- IO_ADC_38 : [IO_Pins.h](#)
- IO_ADC_39 : [IO_Pins.h](#)
- IO_ADC_40 : [IO_Pins.h](#)
- IO_ADC_41 : [IO_Pins.h](#)
- IO_ADC_5V2 : [IO_Pins.h](#)
- IO_ADC_ABSOLUTE : [IO_ADC.h](#)
- IO_ADC_BOARD_TEMP : [IO_Pins.h](#)
- IO_ADC_CURRENT : [IO_ADC.h](#)
- IO_ADC_NODE_ID_0 : [IO_Pins.h](#)
- IO_ADC_NODE_ID_1 : [IO_Pins.h](#)
- IO_ADC_RANGE_10V : [IO_ADC.h](#)
- IO_ADC_RANGE_5V : [IO_ADC.h](#)
- IO_ADC_RATIOMETRIC : [IO_ADC.h](#)
- IO_ADC_RESISTIVE : [IO_ADC.h](#)
- IO_ADC_SENSOR_SUPPLY : [IO_Pins.h](#)
- IO_ADC_UBAT : [IO_Pins.h](#)
- IO_ADC_UBAT_CPU : [IO_Pins.h](#)
- IO_BRBL_CUSTOM_DID_IDX_0 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_1 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_10 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_11 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_12 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_13 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_14 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_15 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_2 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_3 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_4 : [IO_BRBL.h](#)

- IO_BRBL_CUSTOM_DID_IDX_5 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_6 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_7 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_8 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_IDX_9 : [IO_BRBL.h](#)
- IO_BRBL_CUSTOM_DID_TBL_LEN : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_0 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_1 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_10 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_11 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_2 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_3 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_4 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_5 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_6 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_7 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_8 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_IDX_9 : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_LEN : [IO_BRBL.h](#)
- IO_BRBL_XTEA_PRIV_KEY_TBL_LEN : [IO_BRBL.h](#)
- IO_CAN_BAUDRATE_1000K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_100K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_10K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_125K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_20K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_250K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_25K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_500K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_50K : [IO_CAN.h](#)
- IO_CAN_BAUDRATE_800K : [IO_CAN.h](#)
- IO_CAN_CHANNEL_0 : [IO_Pins.h](#)
- IO_CAN_CHANNEL_1 : [IO_Pins.h](#)
- IO_CAN_EXT_FRAME : [IO_CAN.h](#)
- IO_CAN_MSG_READ : [IO_CAN.h](#)
- IO_CAN_MSG_WRITE : [IO_CAN.h](#)
- IO_CAN_STD_FRAME : [IO_CAN.h](#)

- IO_CRYPT_XTEA_KEY_LEN : [IO_Crypt.h](#)
- IO_DI_00 : [IO_Pins.h](#)
- IO_DI_01 : [IO_Pins.h](#)
- IO_DI_02 : [IO_Pins.h](#)
- IO_DI_03 : [IO_Pins.h](#)
- IO_DI_04 : [IO_Pins.h](#)
- IO_DI_05 : [IO_Pins.h](#)
- IO_DI_06 : [IO_Pins.h](#)
- IO_DI_07 : [IO_Pins.h](#)
- IO_DI_10 : [IO_Pins.h](#)
- IO_DI_11 : [IO_Pins.h](#)
- IO_DI_12 : [IO_Pins.h](#)
- IO_DI_13 : [IO_Pins.h](#)
- IO_DI_14 : [IO_Pins.h](#)
- IO_DI_15 : [IO_Pins.h](#)
- IO_DI_16 : [IO_Pins.h](#)
- IO_DI_17 : [IO_Pins.h](#)
- IO_DI_18 : [IO_Pins.h](#)
- IO_DI_19 : [IO_Pins.h](#)
- IO_DI_20 : [IO_Pins.h](#)
- IO_DI_21 : [IO_Pins.h](#)
- IO_DI_22 : [IO_Pins.h](#)
- IO_DI_23 : [IO_Pins.h](#)
- IO_DI_24 : [IO_Pins.h](#)
- IO_DI_25 : [IO_Pins.h](#)
- IO_DI_26 : [IO_Pins.h](#)
- IO_DI_27 : [IO_Pins.h](#)
- IO_DI_28 : [IO_Pins.h](#)
- IO_DI_29 : [IO_Pins.h](#)
- IO_DI_30 : [IO_Pins.h](#)
- IO_DI_31 : [IO_Pins.h](#)
- IO_DI_PD : [IO_DIO.h](#)
- IO_DI_PU : [IO_DIO.h](#)
- IO_DO_00 : [IO_Pins.h](#)
- IO_DO_01 : [IO_Pins.h](#)
- IO_DO_02 : [IO_Pins.h](#)

- IO_DO_03 : [IO_Pins.h](#)
- IO_DO_04 : [IO_Pins.h](#)
- IO_DO_05 : [IO_Pins.h](#)
- IO_DO_06 : [IO_Pins.h](#)
- IO_DO_07 : [IO_Pins.h](#)
- IO_DO_10 : [IO_Pins.h](#)
- IO_DO_11 : [IO_Pins.h](#)
- IO_DO_20 : [IO_Pins.h](#)
- IO_DO_21 : [IO_Pins.h](#)
- IO_DO_22 : [IO_Pins.h](#)
- IO_DO_23 : [IO_Pins.h](#)
- IO_DO_24 : [IO_Pins.h](#)
- IO_DO_25 : [IO_Pins.h](#)
- IO_DO_30 : [IO_Pins.h](#)
- IO_DO_31 : [IO_Pins.h](#)
- IO_DO_32 : [IO_Pins.h](#)
- IO_DO_33 : [IO_Pins.h](#)
- IO_DO_34 : [IO_Pins.h](#)
- IO_DO_35 : [IO_Pins.h](#)
- IO_DRIVER_MODE_DEFAULT : [IO_Driver.h](#)
- IO_DRIVER_MODE_SERVICE_WD : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_NA : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_PORST : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_SW : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_WD : [IO_Driver.h](#)
- IO_DRIVER_RST_STAT_WDT : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_APP : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_AUTH_BL : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RPG_ATTEMPT_NONE : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RSP_NONE : [IO_Driver.h](#)
- IO_DRIVER_RTBM_UDS_RSP_SEND : [IO_Driver.h](#)
- IO_DRIVER_SAFETY_SWITCH_EXT : [IO_Driver.h](#)
- IO_DRIVER_SAFETY_SWITCH_INT : [IO_Driver.h](#)

- IO_DRIVER_SAFETY_SWITCH_NONE : [IO_Driver.h](#)
- IO_DRIVER_SYSTEM_CLOCK : [IO_Driver.h](#)
- IO_E_ADC_CHANNEL_STARTUP : [IO_Constants.h](#)
- IO_E_ADC_INVALID : [IO_Constants.h](#)
- IO_E_BUSY : [IO_Constants.h](#)
- IO_E_CAN_BUS_OFF : [IO_Constants.h](#)
- IO_E_CAN_ERROR_PASSIVE : [IO_Constants.h](#)
- IO_E_CAN_FIFO_FULL : [IO_Constants.h](#)
- IO_E_CAN_INVALID_DATA : [IO_Constants.h](#)
- IO_E_CAN_MAX_HANDLES_REACHED : [IO_Constants.h](#)
- IO_E_CAN_MAX_MO_REACHED : [IO_Constants.h](#)
- IO_E_CAN_OLD_DATA : [IO_Constants.h](#)
- IO_E_CAN_OVERFLOW : [IO_Constants.h](#)
- IO_E_CAN_WRONG_HANDLE : [IO_Constants.h](#)
- IO_E_CH_CAPABILITY : [IO_Constants.h](#)
- IO_E_CHANNEL_BUSY : [IO_Constants.h](#)
- IO_E_CHANNEL_NOT_CONFIGURED : [IO_Constants.h](#)
- IO_E_DI_INVALID_LIMITS : [IO_Constants.h](#)
- IO_E_DI_INVALID_VOLTAGE : [IO_Constants.h](#)
- IO_E_DI_OPEN_LOAD : [IO_Constants.h](#)
- IO_E_DI_OPEN_LOAD_OR_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_DI_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_DI_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_DISCHARGE_FAILED : [IO_Constants.h](#)
- IO_E_DO_CHANNEL_STARTUP : [IO_Constants.h](#)
- IO_E_DO_CURRENT_INACCURATE : [IO_Constants.h](#)
- IO_E_DO_DIAG_TRANSIENT_OSC : [IO_Constants.h](#)
- IO_E_DO_OPEN_LOAD : [IO_Constants.h](#)
- IO_E_DO_OPEN_LOAD_OR_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_DO_OUTPUT_DISABLED : [IO_Constants.h](#)
- IO_E_DO_OUTPUT_STARTUP_ERROR : [IO_Constants.h](#)
- IO_E_DO_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_DO_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_DRIVER_INITIALIZED : [IO_Constants.h](#)

- IO_E_DRIVER_NOT_INITIALIZED : [IO_Constants.h](#)
- IO_E_DRV_SAFETY_CONF_NOT_CONFIG :
[IO_Constants.h](#)
- IO_E_ECU_ALREADY_IN_SAFE_STATE : [IO_Constants.h](#)
- IO_E_EEPROM_BUFFER_FULL : [IO_Constants.h](#)
- IO_E_EEPROM_CRC_MISMATCH : [IO_Constants.h](#)
- IO_E_EEPROM_RANGE : [IO_Constants.h](#)
- IO_E_FET_PROTECTION : [IO_Constants.h](#)
- IO_E_GROUP_CONFLICT : [IO_Constants.h](#)
- IO_E_INVALID_CHANNEL_ID : [IO_Constants.h](#)
- IO_E_INVALID_CRC : [IO_Constants.h](#)
- IO_E_INVALID_DIAG_STATE : [IO_Constants.h](#)
- IO_E_INVALID_PARAMETER : [IO_Constants.h](#)
- IO_E_INVALID_SAFETY_CONFIG : [IO_Constants.h](#)
- IO_E_NO_SAFETY_SWITCH_CONFIGURED :
[IO_Constants.h](#)
- IO_E_NODEID_EEPROM_FALLBACK : [IO_Constants.h](#)
- IO_E_NODEID_EEPROM_INVALID : [IO_Constants.h](#)
- IO_E_NODEID_EEPROM_MISMATCH : [IO_Constants.h](#)
- IO_E_NODEID_PINS_INVALID : [IO_Constants.h](#)
- IO_E_NULL_POINTER : [IO_Constants.h](#)
- IO_E_OK : [IO_Constants.h](#)
- IO_E_PERIODIC_NOT_CONFIGURED : [IO_Constants.h](#)
- IO_E_PID_NO_FREE_HANDLES : [IO_Constants.h](#)
- IO_E_PID_USED : [IO_Constants.h](#)
- IO_E_PROT_ACTIVE : [IO_Constants.h](#)
- IO_E_PROT_FATAL : [IO_Constants.h](#)
- IO_E_PROT_PERMANENT_OFF : [IO_Constants.h](#)
- IO_E_PROT_REENABLE : [IO_Constants.h](#)
- IO_E_PROT_TEMP_OVERLOAD : [IO_Constants.h](#)
- IO_E_PROT_USER_OVERLOAD : [IO_Constants.h](#)
- IO_E_PVG_OUTPUT_DISABLED : [IO_Constants.h](#)
- IO_E_PVG_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_PVG_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_PWD_CAPTURE_ERROR : [IO_Constants.h](#)
- IO_E_PWD_HIGH_LEVEL : [IO_Constants.h](#)

- IO_E_PWD_LOW_LEVEL : [IO_Constants.h](#)
- IO_E_PWD_NOT_FINISHED : [IO_Constants.h](#)
- IO_E_PWD_TIMER_OVERFLOW : [IO_Constants.h](#)
- IO_E_PWM_CAPTURE_ERROR : [IO_Constants.h](#)
- IO_E_PWM_CHANNEL_STARTUP : [IO_Constants.h](#)
- IO_E_PWM_CURRENT_INACCURATE : [IO_Constants.h](#)
- IO_E_PWM_DIAG_TRANSIENT_OSC : [IO_Constants.h](#)
- IO_E_PWM_NOT_FINISHED : [IO_Constants.h](#)
- IO_E_PWM_OPEN_LOAD : [IO_Constants.h](#)
- IO_E_PWM_OPEN_LOAD_OR_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_PWM_OUTPUT_DISABLED : [IO_Constants.h](#)
- IO_E_PWM_OUTPUT_HIGH : [IO_Constants.h](#)
- IO_E_PWM_OUTPUT_LOW : [IO_Constants.h](#)
- IO_E_PWM_OUTPUT_STARTUP_ERROR : [IO_Constants.h](#)
- IO_E_PWM_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_PWM_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_RESET_COUNTER_INVALID : [IO_Constants.h](#)
- IO_E_SAFETY_NOT_SUPPORTED : [IO_Constants.h](#)
- IO_E_SBRAM_CONTENT_INVALID : [IO_Constants.h](#)
- IO_E_SPI_BUFFER_FULL : [IO_Constants.h](#)
- IO_E_SPI_MAX_DEV_REACHED : [IO_Constants.h](#)
- IO_E_SW_INTERNAL : [IO_Constants.h](#)
- IO_E_SW_OUTPROT_SM : [IO_Constants.h](#)
- IO_E_TASK_NO_FREE_SLOTS : [IO_Constants.h](#)
- IO_E_UART_BUFFER_EMPTY : [IO_Constants.h](#)
- IO_E_UART_BUFFER_FULL : [IO_Constants.h](#)
- IO_E_UART_OVERFLOW : [IO_Constants.h](#)
- IO_E_UART_PARITY : [IO_Constants.h](#)
- IO_E_UNKNOWN : [IO_Constants.h](#)
- IO_E_VOUT_OUTPUT_DISABLED : [IO_Constants.h](#)
- IO_E_VOUT_PRECISION : [IO_Constants.h](#)
- IO_E_VOUT_SHORT_BATTERY : [IO_Constants.h](#)
- IO_E_VOUT_SHORT_CIRCUIT : [IO_Constants.h](#)
- IO_E_WD_INT_ONLY_NON_SAFETY : [IO_Constants.h](#)
- IO_E_WD_TRIGGER_DISABLED : [IO_Constants.h](#)

- IO_E_WD_TRIGGER_TEMPORARY_DISABLED : [IO_Constants.h](#)
- IO_E_WRONG_HW_TYPE : [IO_Constants.h](#)
- IO_INT_POWERSTAGE_ENABLE : [IO_Pins.h](#)
- IO_INT_PVG_VOUT_0_ENABLE : [IO_Pins.h](#)
- IO_INT_PVG_VOUT_1_ENABLE : [IO_Pins.h](#)
- IO_K15 : [IO_Pins.h](#)
- IO_LED_00 : [IO_Pins.h](#)
- IO_LED_01 : [IO_Pins.h](#)
- IO_LED_02 : [IO_Pins.h](#)
- IO_LED_03 : [IO_Pins.h](#)
- IO_LED_04 : [IO_Pins.h](#)
- IO_LED_05 : [IO_Pins.h](#)
- IO_LED_06 : [IO_Pins.h](#)
- IO_LED_07 : [IO_Pins.h](#)
- IO_PID_MAX_HANDLES : [IO_PID.h](#)
- IO_POWER_OFF : [IO_POWER.h](#)
- IO_POWER_ON : [IO_POWER.h](#)
- IO_PVG_00 : [IO_Pins.h](#)
- IO_PVG_01 : [IO_Pins.h](#)
- IO_PVG_02 : [IO_Pins.h](#)
- IO_PVG_03 : [IO_Pins.h](#)
- IO_PVG_04 : [IO_Pins.h](#)
- IO_PVG_05 : [IO_Pins.h](#)
- IO_PWD_00 : [IO_Pins.h](#)
- IO_PWD_01 : [IO_Pins.h](#)
- IO_PWD_02 : [IO_Pins.h](#)
- IO_PWD_03 : [IO_Pins.h](#)
- IO_PWD_10 : [IO_Pins.h](#)
- IO_PWD_11 : [IO_Pins.h](#)
- IO_PWD_12 : [IO_Pins.h](#)
- IO_PWD_13 : [IO_Pins.h](#)
- IO_PWD_20 : [IO_Pins.h](#)
- IO_PWD_21 : [IO_Pins.h](#)
- IO_PWD_22 : [IO_Pins.h](#)
- IO_PWD_23 : [IO_Pins.h](#)

- IO_PWD_BOTH_COUNT : [IO_PWD.h](#)
- IO_PWD_DOWN_COUNT : [IO_PWD.h](#)
- IO_PWD_FALLING_COUNT : [IO_PWD.h](#)
- IO_PWD_FALLING_VAR : [IO_PWD.h](#)
- IO_PWD_HIGH_TIME : [IO_PWD.h](#)
- IO_PWD_INC_1_COUNT : [IO_PWD.h](#)
- IO_PWD_INC_2_COUNT : [IO_PWD.h](#)
- IO_PWD_LOW_TIME : [IO_PWD.h](#)
- IO_PWD_PD : [IO_PWD.h](#)
- IO_PWD_PERIOD_TIME : [IO_PWD.h](#)
- IO_PWD_PU : [IO_PWD.h](#)
- IO_PWD_RESOLUTION_0_2 : [IO_PWD.h](#)
- IO_PWD_RESOLUTION_0_4 : [IO_PWD.h](#)
- IO_PWD_RESOLUTION_0_8 : [IO_PWD.h](#)
- IO_PWD_RESOLUTION_1_6 : [IO_PWD.h](#)
- IO_PWD_RESOLUTION_3_2 : [IO_PWD.h](#)
- IO_PWD_RISING_COUNT : [IO_PWD.h](#)
- IO_PWD_RISING_VAR : [IO_PWD.h](#)
- IO_PWD_UP_COUNT : [IO_PWD.h](#)
- IO_PWM_00 : [IO_Pins.h](#)
- IO_PWM_01 : [IO_Pins.h](#)
- IO_PWM_02 : [IO_Pins.h](#)
- IO_PWM_03 : [IO_Pins.h](#)
- IO_PWM_04 : [IO_Pins.h](#)
- IO_PWM_05 : [IO_Pins.h](#)
- IO_PWM_10 : [IO_Pins.h](#)
- IO_PWM_11 : [IO_Pins.h](#)
- IO_PWM_CURRENT_QUEUE_MAX : [IO_PWM.h](#)
- IO_SAFETY_SWITCH_0 : [IO_Pins.h](#)
- IO_SAFETY_SWITCH_1 : [IO_Pins.h](#)
- IO_SAFETY_SWITCH_NONE : [IO_Driver.h](#)
- IO_UART : [IO_Pins.h](#)
- IO_VOUT_00 : [IO_Pins.h](#)
- IO_VOUT_01 : [IO_Pins.h](#)
- IO_VOUT_02 : [IO_Pins.h](#)
- IO_VOUT_03 : [IO_Pins.h](#)

- IO_VOUT_04 : [IO_Pins.h](#)
- IO_VOUT_05 : [IO_Pins.h](#)

Generated on Mon Nov 16 2020 16:59:49 for HY-TTC 30 Family C API Manual by

 1.8.2

HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page		Related Pages		Data Structures		Files	
File List		Globals					
All	Functions	Typedefs	Enumerations	Enumerator			
Macros							
a	d	f	i	n	t	u	

- n -

- NULL : [ptypes_xe167.h](#)

HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page		Related Pages		Data Structures		Files	
File List		Globals					
All	Functions	Typedefs	Enumerations	Enumerator			
Macros							
a	d	f	i	n	t	u	

- t -

- TRUE : [ptypes_xe167.h](#)

HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page		Related Pages		Data Structures		Files	
File List		Globals					
All	Functions	Typedefs	Enumerations		Enumerator		
Macros							
a	d	f	i	n	t	u	

- u -

- UINT32_ALL_BITS_SET : [TypesGen.h](#)

[Main Page](#)[Related Pages](#)[Data Structures](#)[Files](#)

Related Pages

Here is a list of all related documentation pages:

Diagnostic state machine error codes

Details about the errors of the diagnostic state machine

HY-TTC30 Family pin features

Listing of all IO driver pins and their configuration options

Pin and diagnostic features

Explicit overview of the diagnostic functions of the ECU pins

ECU Map

Description and properties of the ECU

Implementation Examples for Safety Functions

Example for
using the IO-
Driver in a
safety
critical
environment

Examples for using UDS support functions

Generated on Mon Nov 16 2020 16:59:48 for HY-TTC 30 Family C API Manual by

doxygen 1.8.2



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page	Related Pages	Data Structures	Files	
External				

External Directory Reference

Directories

directory **LogisticTypes**

Generated on Mon Nov 16 2020 16:59:48 for HY-TTC 30 Family C API Manual by

doxygen 1.8.2



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page	Related Pages	Data Structures	Files
External	LogisticTypes		

LogisticTypes Directory Reference

Directories

directory **Apdb**

directory **Types**

Generated on Mon Nov 16 2020 16:59:48 for HY-TTC 30 Family C API Manual by

doxygen 1.8.2



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page	Related Pages	Data Structures	Files
External	LogisticTypes	Apdb	

Apdb Directory Reference

Files

file **Apdb.h**
APDB define for bootloader.

[Main Page](#)[Related Pages](#)[Data Structures](#)[Files](#)[inc](#)

inc Directory Reference

Files

file **ApdbCfg.h**
Definitions for **Apdb.h**.

file **DIAG_Constants.h**
Global defines for IO Driver diagnostic state machine and WD.

file **DIAG_Functions.h**
Auxiliary functions for the diagnostic state machine.

file **IO_ADC.h**
IO Driver functions for ADC.

file **IO_BRBL.h**
API for accessing data in the branding block of the ECU.

file **IO_CAN.h**
IO Driver functions for CAN communication.

file **IO_Constants.h**

Global defines for IO Driver.

file **IO_Crypt.h**

API for I/O driver cryptographic functions.

file **IO_DIO.h**

IO Driver functions for Digital Input/Output.

file **IO_Driver.h**

High level interface to IO Driver.

file **IO_EEPROM.h**

IO Driver functions for EEPROM.

file **IO_EEPROM_Preload.h**

Pre-load functions for EEPROM.

file **IO_LED.h**

IO driver functions for LED.

file **IO_NodeID.h**

IO Driver functions for reading the NodeID pins.

file **IO_PID.h**

Contains the data structure for configuring the PID controller.

file **IO_Pins.h**

Global IO Pin defines for IO Driver.

file **IO_POWER.h**

IO Driver functions for Power control.

file **IO_PVG.h**

IO Driver functions for PVG channels.

file **IO_PWD.h**

IO Driver functions for timer input channels.

file **IO_PWM.h**

IO Driver functions for PWM channels.

file **IO_RTC.h**

RTC functions, provides exact timing functions.

file **IO_UART.h**

IO Driver functions for UART communication.

file **IO_Vout.h**

IO Driver functions for voltage outputs.

file **IO_WD.h**

IO-Driver for the Window Watchdog.

file **IO_WDTimer.h**

IO Driver functions for the CPU's Watchdog timer.

file **ptypes_xe167.h**

Primitive data types.



HY-TTC 30 Family C API Manual D-TTC-X-G- 20-001

Main Page	Related Pages	Data Structures	Files
External	LogisticTypes	Types	

Types Directory Reference

Files

file **TypesGen.h**
Types header file.