

Software API
API 200

August 28, 2008





REVISION HISTORY

The following major modifications and improvements have been made to the first version of this document:

No	Major Changes
0.96	Corrected: No internal pull-up on IN_5
1.00	Product released

**Published by EnOcean GmbH, Kolpingring 18a, 82041 Oberhaching, Germany
 www.enocean.com, info@enocean.com, phone ++49 (89) 6734 6890**

© EnOcean GmbH
 All Rights Reserved

Important!

This information describes the type of component and shall not be considered as assured characteristics. No responsibility is assumed for possible omissions or inaccuracies. Circuitry and specifications are subject to change without notice. For the latest product specifications, refer to the EnOcean website: <http://www.enocean.com>.

As far as patents or other rights of third parties are concerned, liability is only assumed for modules, not for the described applications, processes and circuits.

EnOcean does not assume responsibility for use of modules described and limits its liability to the replacement of modules determined to be defective due to workmanship. Devices or systems containing RF components must meet the essential requirements of the local legal authorities.

The modules must not be used in any relation with equipment that supports, directly or indirectly, human health or life or with applications that can result in danger for people, animals or real value.

Components of the modules are considered and should be disposed of as hazardous waste. Local government regulations are to be observed.

Packing: Please use the recycling operators known to you. By agreement we will take packing material back if it is sorted. You must bear the costs of transport. For packing material that is returned to us unsorted or that we are not obliged to accept, we shall have to invoice you for any costs incurred.

TABLE OF CONTENT

1	GENERAL DESCRIPTION.....	5
1.1	Basic Functionalities	5
1.2	Features Overview	6
1.3	Ordering Information.....	6
2	FUNCTIONAL DESCRIPTION.....	7
2.1	Block Diagram.....	7
2.2	Pin Description	7
3	APPLICATIONS INFORMATION	8
4	API Description.....	9
	# define _CONFIG.....	9
	# define _NR_RX_RADIO_BUFFERS	9
	# define _NR_TX_RADIO_BUFFERS	9
	char initAPI200(void)	10
	char sendSerialByte(unsigned char value).....	10
	char getSerialByte(unsigned char *value)	10
	char sendSerialTelegram(union telegram *telegram)	12
	char getRadioTelegram(union telegram *telegram)	17
	char sendRadioTelegram(union telegram *telegram)	18
	char setRadioFilter(unsigned long ID).....	18
	char setRadioBufferMaturity(unsigned char time)	18
	char setSerialSpeed(unsigned char SpeedCode).....	19
	char clearRadioFilter().....	19
	char setIDBase(unsigned long IDBase).....	19
	unsigned long readIDBase().....	19
	char setRxSensitivity(unsigned char sensitivity)	20
	char readRxSensitivity()	20
	char sleep(unsigned long time)	20
	char receiverOn()	21
	char receiverOff()	21
	char repeaterOn()	21
	char repeaterOff()	21
	void reset()	22
	char writeOut(unsigned char outPin, unsigned char level)	22
	char configInputs(unsigned char configCode)	22
	char configSerialPins(unsigned char pinConfigCode)	23
	char readADIn(unsigned char inPin, unsigned char resolution, unsigned int *ADconversion)	23
	char readDigitalIn(unsigned char inPin, unsigned char *level)	24
	unsigned long getTime()	24
	void nop()	24
	void wait(unsigned long delay)	24
	void clrWdt()	25
	void switchWdtOn()	25
	void switchWdtOff()	25
	char writeFlash()	25
	char readFlash().....	26
	unsigned char getNoiseLevel()	26



A. Development Tools27
 A.1 Microchip Tools27
 A.2 EnOcean Tools28
B. Installation.....30
C. Sample Applications31
 C.1 Default Application31
 C.1 Function to show noise level with 3 LEDs32

1 GENERAL DESCRIPTION

The API 200 is a software API for the transceiver modules TCM200C/TCM220C of EnOcean. It enables the realization of bi-directional RF applications based on the innovative EnOcean radio technology.

The TCM200C/TCM220C transceiver module serves the 315 MHz air interface protocol of EnOcean. It receives all signals of the EnOcean radio transmitters and makes them available via an API. The API provides several hardware interfaces such as a serial interface, 10bit A/D inputs, digital inputs and digital outputs.

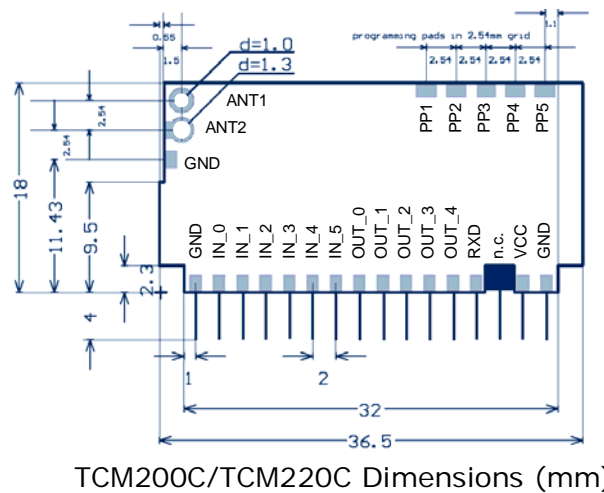


Figure 1: Transceiver module TCM200C/TCM220C

1.1 Basic Functionalities

- Receive and transmit EnOcean radio telegrams
 - All different kinds of EnOcean telegrams (e.g. PTM200C or STM110C) can be generated
 - The API 200 can send on 128 different IDs (derived from 1 base number)
 - The API 200 provides a radio filter functionality
- Serial interface support
- Read analog and digital inputs, write digital outputs
- FLASH read/write
- Timer functionality
- Various power-down modes

1.2 Features Overview

Radio transmission and reception	according to EnOcean radio protocol
Sleep current	TCM 200: typ. 1.5mA, TCM220C: typ. 50 μ A ¹
Current consumption (receiver on/off)	typ. 29 mA (max. 34 mA) mA ¹
Current consumption (transmit)	max. 40 mA ¹
Receiver features	sensitivity can be reduced by SW command
Bi-directional serial interface	full-duplex, async., 9.6 - 57.6 kbps
Inputs	up to 6 inputs, configurable as digital or analog input; AD-conversion up to 10bit
Outputs	up to 5 digital outputs
Free Memory	256 bytes data in FLASH, ~16 kB program memory in FLASH ² , ~0.5 kB RAM ²

1.3 Ordering Information

Type	Ordering Code
TCM 200C	S3033-K200
TCM 220C	S3033-K220
EDK 100C	S3034-X100



The programming port connector, which is mounted for development purposes on the modules supplied with the development kit EDK100C is not mounted on standard TCM 200C modules. In order to program the TCM 200C/220C modules with own software in series production it is necessary to contact the programming pins with needles.

¹ All pins except Vcc / GND not connected at the time of measurement. All outputs set to 0.

² API functions are linked to the program code as needed. The more functions are used the higher is the memory area consumed by the API.

2 FUNCTIONAL DESCRIPTION

2.1 Block Diagram

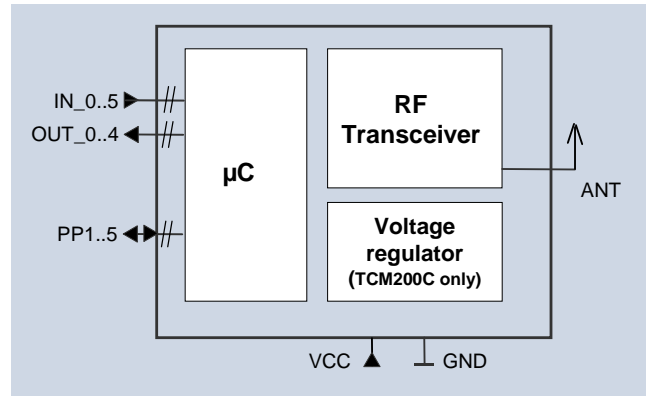


Figure 2: Block diagram of TCM200C / TCM220C

2.2 Pin Description

Pin	Symbol	Function TCM200C	Function TCM220C
1	GND	Ground connection	
2	IN_0	Digital inputs 3V logic, 5V tolerant, internal pull up (3V)	Digital inputs, 3V logic or analog inputs 3V
3	IN_1		
4	IN_2		
5	IN_3		
6	IN_4		
7	IN_5/SER_RX	Digital input or serial receive line, 3V logic, 5V tolerant	Digital input or serial receive line, 3V logic
8	OUT_0/SER_TX	Open drain output or serial transmit line (OUT_0/SER_TX), 35 V max., 100 mA max., 100 mW max. each.	Digital output or serial transmit line (OUT_0/SER_TX), 3V logic, 20mA max.
9	OUT_1		
10	OUT_2		
11	OUT_3		
12	OUT_4	Digital output, 5V logic, 20mA max.	Digital output, 3V logic, 20mA max.
13	RXD	For EnOcean internal use only	
14	n.c.	Not used	
15	VCC	Power supply 5V ± 5%	Power supply 3V ± 5%
16	GND	Ground connection	
	ANT1	Foot point for whip antenna	
	ANT2	Foot point for 50Ω antenna	
PP1	ICSP_VPP	Programming voltage or active low reset to controller	
PP2	Vcc	3V internal VCC for programming interface	
PP3	GND	Ground connection for programming interface	
PP4	ICSP_DATA	In-circuit debugger and ICSP programming data	
PP5	ICSP_CLK	In-circuit debugger and ICSP programming clock	

3 APPLICATIONS INFORMATION

Please refer to the user manual of the TCM200C/TCM220C hardware!

4 API Description



Please note that all timer interrupts are occupied by the API. If interrupts are used in the customer application the API may not work properly!

define _CONFIG

The first source code line of user application code must contain

```
#define _CONFIG Code
```

Through this define the user configures the watch dog timer (WDT) period. The following table explains the relation between the code and the configuration.

Code	WDT approximate period (ms)
1	16
2	128
3	2048



The WDT period strongly depends on temperature. Please refer to the data sheet of the PIC18F65J11 micro controller for more details. If not specified _CONFIG 3 is used as default configuration.

define _NR_RX_RADIO_BUFFERS

With this define the number of rx radio buffers is defined:

```
# define _NR_RX_RADIO_BUFFERS NumBuf
```

NumBuf must be between 4 and 10. The default value is 4.



- The free RAM size is reduced by increasing the number of radio buffers.
- The number of radio buffers must be defined before *initAPI200()* is executed.

define _NR_TX_RADIO_BUFFERS

With this define the number of tx radio buffers is defined:

```
# define _NR_TX_RADIO_BUFFERS NumBuf
```

NumBuf must be between 1 and 11. The default value is 1.



- The free RAM size is reduced by increasing the number of radio buffers.
- The number of radio buffers must be defined before *initAPI200()* is called
- If the repeater functionality is used the number of TX radio buffers must be at least the number of RX radio buffers + 1

char initAPI200(void)

This function must be inserted as first line code in the main program. Initializes the internal registers. The startup procedure of the module takes about 85 ms (due to micro controller).

Returns

- 0 No error. API200 correctly initialized
- 1 Radio buffer wrong initialized: RX buffer address, amount of RX radio buffer out of specified value. API200 not correctly initialized.
- 2 Reset happened due to stack overflow
- 3 Reset happened due to stack underflow
- 4 Reset happened due to Brown-out detection
- 5 Reset happened due to watch dog timer overflow
- 6 Reset happened due to power-down
- 7 Radio buffer wrong initialized: TX buffer address, amount of TX radio buffer out of specified value. API200 not correctly initialized.

Status after execution of initAPI200():

- Radio reception deactivated. Use receiverOn() to activate the radio reception.
- Serial port activated: Pin 7 configured as SER_RX. Pin 8 Configured as SER_TX. Use configSerialPins() to configure the pins as I/O.
- The rest of the output pins low, or not conducting if controlled by an output transistor. Use writeOut() to change the state.
- Maturity time 100ms. Use setRadioBufferMaturity() to select a different maturity time.
- Repeater off. Use repeaterOn() to activate the repeater.
- Radio rx sensitivity high. Use setRxSensitivity() to set a low sensitivity.
- Watchdog timer on. Use switchWdtOff() to deactivate the WDT.
- TCM220C inputs configured as digital. Use configInputs() to configure inputs as analogue.

char sendSerialByte(unsigned char value)

Send one byte via the serial interface. There is a buffer for 126 bytes.

Returns

- 0 byte sent
- 1 information could not be placed in sender buffer within 20ms
- 2 SER_RX/TX configured as digital input/output

char getSerialByte(unsigned char *value)

If a byte has been received via the serial interface it can be read with the *getSerialByte()* function. There is a buffer for 14 bytes.



The radio receiver routine has priority over the serial interface. In case of radio communication incoming serial bytes may be lost. An error handling should be provided by the application.

Returns

0	byte received
1	no byte received
2	SER_RX/TX configured as digital output/input
3	USART RX buffer overrun
4	Framing error. Maybe due to serial speed mismatch. Will be cleared automatically after correct reception of one byte.
11	parameter address not allowed

char sendSerialTelegram(union telegram *telegram)

The function is used to log the content of received or transmitted telegrams via the serial interface or to transmit other information via the serial interface. The information is sent according to the EnOcean serial port protocol. There are several possibilities to address the bytes in this structure as shown below. There is a buffer for 9 telegrams (126 bytes).



The transmission of a serial telegram takes at least about 15ms. Please take care not to enter sleep mode before the transmission of the previous telegram has been finished.

```
union telegram{
    struct{
        unsigned char type;
        unsigned char org;
        unsigned char data3;
        unsigned char data2;
        unsigned char data1;
        unsigned char data0;
        unsigned char id3;
        unsigned char id2;
        unsigned char id1;
        unsigned char id0;
        unsigned char status;
    };
    struct{
        unsigned char type;
        unsigned char org;
        unsigned char mdata5;
        unsigned char mdata4;
        unsigned char mdata3;
        unsigned char mdata2;
        unsigned char mdata1;
        unsigned char mdata0;
        unsigned char mid1;
        unsigned char mid0;
        unsigned char status;
    };
    struct{
        unsigned char type;
        unsigned char org;
        unsigned char : 8;
        unsigned char : 8;
        unsigned char : 8;
        unsigned char : 8;
        unsigned char aid1;
        unsigned char aid0;
        unsigned char : 8;
        unsigned char : 8;
        unsigned char status;
    };
};
```

Returns

- 0 telegram sent
- 1 information could not be placed in sender buffer within 20ms
- 2 SER_RX/TX configured as digital input/output
- 11 parameter address not allowed

Message format

The following figure shows the message format. A data block of length n is composed of 2 synchronization bytes, 1 octet for the header and n-1 octets for the message data.

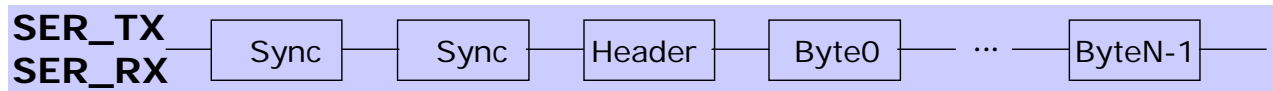


Figure 3: Message format for asynchronous serial communication

Octet signals and bit order

- Default 9.6 kbps, 8 data bits, no parity bit, one start bit, one stop bit
- Line idle is binary 1 (standard)
- Each character has one start bit (binary 0), 8 information bits (least significant bit first) and one stop bit (binary 1)

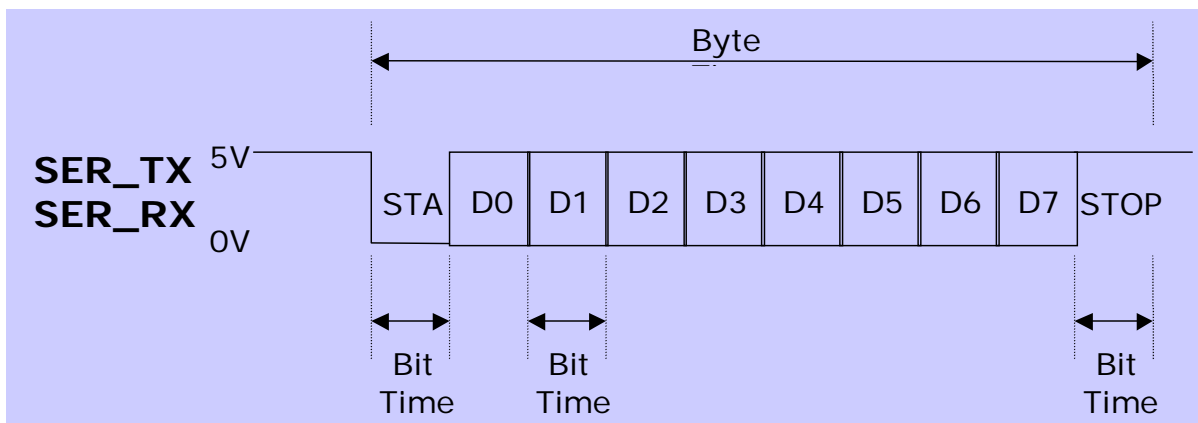


Figure 4: Signals and bit order sending a byte

General encoding	Encoding for RPS, 1BS, 4BS, HRC	Encoding for 6DT	Encoding for MDA
0xA5 (sync byte)	0xA5 (sync byte)	0xA5 (sync byte)	0xA5 (sync byte)
0x5A (sync byte)	0x5A (sync byte)	0x5A (sync byte)	0x5A (sync byte)
type	0x0B (TX_TELEGRAM) ¹ 0x6B(RX_TELEGRAM)	0x0B (TX_TELEGRAM) 0x6B(RX_TELEGRAM)	0x0B (TX_TELEGRAM) 0x6B(RX_TELEGRAM)
org	org	0x0A	0x0B
data3	data3	mdata5	0xXX
data2	data2	mdata4	0xXX
data1	data1	mdata3	0xXX
data0	data0	mdata2	0xXX
id3	id3	mdata1	aid1
id2	id2	mdata0	aid0
id1	id1	mid1	0xXX
id0	id0	mid0	0xXX
status	status	status	status
ChkSum	ChkSum	ChkSum	ChkSum

Remarks

- type: value 0x0B and 0x6B may be used, other values reserved for future
- data2 = data1 = data0 = 0x00 for RPS,1BS, HRC

Detailed description of ORG field

ORG	Description	RRT / TRT Acronym
0x05	Telegram from a PTM switch module received (original or repeated message)	RPS
0x06	1 byte data telegram from a STM sensor module received	1BS
0x07	4 byte data telegram from a STM sensor module received	4BS
0x08	Telegram from a CTM module received	HRC
0x0A	6byte Modem Telegram	6DT
0x0B	Modem acknowledge telegram	MDA

Description of data0..3 and id0..3

If ORG = 0x05 and NU = 1 (N-message from a PTM switch module)

id3..0 32bit transmitter ID
 data2..0 always = 0
 data3 as follows:

¹ viewed from TCM module



API 200

7				0	
RID	UD	PR	SRID	SUD	SA

RID (2 bit) Rocker ID, from left (A) to right (D): 0, 1, 2 and 3 (decimal)
 UD (1 bit) UD=1 → O-button, UD=0 → I-button
 PR (1 bit) PR=1 → Button pressed, PR=0 → Button released
 SRID (2 bit) Second Rocker ID, from left to right: 0, 1, 2 and 3
 SUD (1 bit) (Second) SUD=1 → Up button, SUD=0 → Down button
 SA (1 bit) SA=1 → Second action, SA=0 → No second action

If ORG = 0x05 and NU = 0 (U-message from a PTM switch module)

id3..0 32bit transmitter ID
 data2..0 always = 0
 data3 as follows:

7		0
BUTTONS	PR	Reserved

BUTTONS (3 bit) Number of simultaneously pressed buttons, as following:

	PTM 100 (Type1):	PTM200 (Type2):
	0 = 0 Buttons	0 = 0 Button
	1 = 2 Buttons	1 = not possible
	2 = 3 Buttons	2 = not possible
	3 = 4 Buttons	3 = 3 or 4 buttons
	4 = 5 Buttons	4 = not possible
	5 = 6 Buttons	5 = not possible
	6 = 7 Buttons	6 = not possible
	7 = 8 Buttons	7 = not possible

PR (1 bit) PR = 1 → Button pressed, PR = 0 → Button released
 Reserved (4 bit) for future use

If ORG = 0x06 (Telegram from a 1 Byte STM sensor)

id3..0 32bit transmitter ID
 data2..0 always = 0
 data3 Sensor data byte

If ORG = 0x07 (Telegram from a 4 Byte STM sensor)

id3..0 32bit transmitter ID
 data3 Value of third sensor analog input
 data2 Value of second sensor analog input
 data1 Value of first sensor analog input
 data0 Sensor digital inputs as follows:

PTM switch modules of Type 1 (e.g. PTM 100) do not support interpretation of operating more than one rocker at the same time:

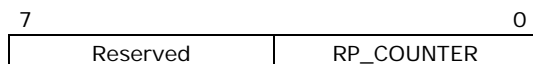
- N-message received → Only one pushbutton was pressed.
- U-message received → No pushbutton was pressed when activating the energy generator, or more than one pushbutton was pressed.

PTM switch modules of Type 2 allow interpretation of operating two buttons simultaneously:

- N-message received → Only one or two pushbuttons have been pressed.
- U-message received → No pushbutton was pressed when activating the energy generator, or more than two pushbuttons have been pressed.

Note for telegrams from PTM 100 piezo transmitters: Due to the mechanical hysteresis of the piezo energy bow, in most rocker switch device implementations, pressing the rocker sends an N-message and releasing the rocker sends a U-message!

If ORG = 0x06, 0x07, 0x08 or 0x0A (all other telegrams)



Reserved	(4 bit)	for future use
RP_COUNTER	(4 bit)	Repeater level: 0 original message 1 repeated message

Description of ChkSum

Least Significant Byte from addition of all octets except sync bytes and checksum.
The checksum is calculated automatically by the library.

char getRadioTelegram(union telegram *telegram)

When a radio telegram is received the telegram content (11 byte) is stored internally. There is a buffer for several telegrams. The number of buffers is defined via # define `_NR_RX_RADIO_BUFFERS` as described above. The default number of buffers is 4 (maximum value 10). With the function `getRadioTelegram()` the telegram content is made available. Identical (including repeated) telegrams arriving within the maturity time of the buffer (default 100ms) are treated as one telegram (typically EnOcean transmitters are repeating the same message 3 times to improve the transmission probability).

Returns

- 0 telegram received
- 1 telegram filtered.
- 2 not received telegram
- 3 telegram stored in the last free rx radio buffer
- 4 rx radio buffer overflow.
- 11 parameter address not allowed

char sendRadioTelegram(union telegram *telegram)

With this command one single radio telegram can be transmitted. In order to achieve a high level of transmission probability it is recommended to send 3 telegrams at random intervals within 40ms. Random intervals can be generated using the pseudo random functions in `stdlib.h` of the C18 compiler.



At high signal levels the receiver in TCM2xxC needs the first telegram to adjust the automatic gain control. This telegram cannot be received and is lost. Therefore a minimum of two telegrams should be transmitted!

Returns

- 0 telegram sent
- 1 ID incorrect or data not allowed. Telegram not sent.
- 2 ORG byte not allowed. Telegram not sent
- 3 telegram couldn't be sent within 5ms
- 4 telegram written in last free tx radio buffer
- 11 parameter address not allowed

char setRadioFilter(unsigned long ID)

All received telegrams except modem telegrams will be filtered and all telegrams which do not stem from the module with the specified ID will be discarded.



Filtering is done only for the `getRadioTelegram()` function. The repeater – if activated - will repeat also telegrams with other IDs.

Returns

- 0 filter set

char setRadioBufferMaturity(unsigned char time)

With this command the maturity time of the rx radio buffers may be changed. The default value is 100ms. Identical telegrams arriving within the maturity time are treated as one telegram. This is necessary because a transmitter usually transmits up to 3 sub-telegrams. In addition a repeater may increase this number. 100ms are set as default value for optimal performance with a repeater. If no repeater is installed or the transmitter transmits less sub-telegrams the maturity time can be reduced. This may be helpful in environments with a large number of transmitters.

The value for *time* must be between 2 and 254 (ms).

Returns:

- 0 Maturity value correctly written.
- 1 Maturity value correctly written. Value under 100ms.
- 2 Error code. Maturity value not written. Input value <2ms or >254ms
- 3 Error code. Maturity value not written. Value under 60ms not allowed when repeater active

char setSerialSpeed(unsigned char SpeedCode)

With this command the serial speed may be set. The default value is 9600 bit/s.

Code	Serial speed (bit/s)
1	9600
2	19200
3	38400
4	57600

Returns:

- 0 Serial speed correctly set
- 1 Error code. Invalid input value.

char clearRadioFilter()

No radio telegram IDs are filtered

Returns

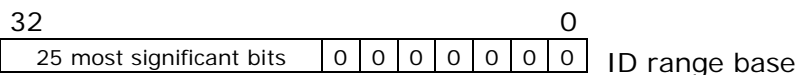
- 0 filter cleared

char setIDBase(unsigned long IDBase)

With this command the user can rewrite its ID range base number. The information of the 25 most significant bits is stored in FLASH. This command can only be used a maximum number of 10 times to avoid misuse.



It takes about 50ms to write to FLASH. If a watch dog timer period shorter than that is used the watch dog timer has to be switched off before execution of this command to avoid a reset!



Returns

- 0 new ID base set
- 1 ID base < 0xFF800000. ID base not allowed. New ID not set
- 2 ID programmed more than 10 times. New ID not set
- 3 Error when trying writing to FLASH



The default value of the ID after programming is "0xFFFFFFFF80". Therefore the ID has to be changed after programming. It is recommended to generate a random number in the allowed address range and write it via setIDBase().

unsigned long readIDBase()

Returns

unsigned long IDBase;

Codes in [0xFF800000, 0xFFFFFFFF] with 7 LSBs to 0

char setRxSensitivity(unsigned char sensitivity)

With this command the TCM radio sensitivity is set. In LOW radio sensitivity signals from far transmitters are not detected by the TCM receiver. This feature is useful when only information from transmitters in the vicinity should be processed.

sensitivity: !0 = high sensitivity
 0 = low sensitivity

Returns

0 sensitivity set

char readRxSensitivity()

Returns

0 low radio sensitivity (typ. -65 dBm)
 !0 high radio sensitivity (typ. -95 dBm)

char sleep(unsigned long time)

Sleep for a time interval specified in WDT period units (as defined in _CONFIG). The I/O ports maintain the status they had before the sleep instruction was executed. For lowest current consumption in this mode, place all I/O pins at either VDD or VSS, ensure no external circuitry is drawing current from the I/O pin. Pull all I/O pins that are hi-impedance inputs, high or low externally, to avoid switching currents caused by floating inputs.

With a rising edge on the SER_RX line it is also possible to wake up the module.



- If the WDT has been switched off the unit will only wake up with a rising edge on SER_RX!
- If a telegram has been sent via the serial interface using the sendSerialTelegram() command it is necessary to wait for about 20ms before entering the sleep mode. Otherwise the serial telegram will not be transmitted completely.
- RX/TX radio buffers are erased by the function before entering in sleep.

Returns:

0 Watch dog timer woke up μ C from sleep
 1 External signal woke up μ C from sleep
 2 Communication problem with Chipcon. System did not enter sleep. Retry.
 3, 4 System could not reestablish correctly the Chipcon configurations after sleep

char receiverOn()

The receiver is switched off after the start-up of the module. With this command the receiver is switched on, the power consumption strongly increases.

```
extern unsigned char rec_pdown; /* TRUE if receiver powered down,  
                                FALSE if receiver is running */
```

The flag `rec_pdown` is set according to the current status.

Returns

0 radio receiver enabled

char receiverOff()

With this function the receiver can be switched off. The flag `rec_pdown` is set according to the current status.

```
extern unsigned char rec_pdown; /* TRUE if receiver powered down,  
                                FALSE if receiver is running */
```

Returns

0 radio receiver disabled

char repeaterOn()

Switches the integrated repeater function on. For a set of sub-telegrams belonging to the same telegram the repeater functions generates typically three repeated sub-telegrams. If the air channel is occupied at the moment of sending the number of repeated sub-telegrams may be smaller than 3. Telegrams which have already been repeated are not repeated again. Remote learn telegrams are not repeated!



The *receiverOn()* command is executed automatically. The *getRadioTelegram()* routine must be called on a regular basis! The RX buffers are only released if the maturity time has elapsed and the buffer content has been read via *getRadioTelegram()*. Otherwise the repeater will stop working!

Returns

0 radio repeater enabled
1 Maturity time < 60ms radio. Repeater could not be enabled.
2 Number of TX radio buffers < (Number of RX radio buffers+1)
3 radio receiver control pin (RCO) wrongly configured as input. Action not performed

char repeaterOff()

Switches the integrated repeater function off. The radio reception remains active. The RX and TX buffers will be cleared.

Returns

0 radio repeater disabled

void reset()

Performs a reset of the module.

char writeOut(unsigned char outPin, unsigned char level)

Set OUT_0 to OUT_4 level as defined in level. The current status is stored in the global structure output.

```
char outPin; /* 0 to 4 for OUT_0 to OUT_4 */
char level; /* 0: Digital output LOW or output transistor (TCM200C) non-conducting
            1: Digital output HIGH or output transistor (TCM200C) conducting */
```

```
extern struct out{
unsigned OUT0    :1;
unsigned OUT1    :1;
unsigned OUT2    :1;
unsigned OUT3    :1;
unsigned OUT4    :1;
unsigned reserved :3;
}output;
```

Returns

```
0 state written to pin
1 some output pin is configured as input
2 pin value not allowed
3 OUT_0/SER_TX configured as SER_TX
```

char configInputs(unsigned char configCode)

This function configures the input pins as analog, digital or reference input. The table summarizes the available codes and configurations. The default configuration is 0x00. If the SER_RX is configured as IN_4 (configSerialPins) it is always a digital input.



This function may only be used with TCM220C hardware! TCM200C does not provide analog inputs!

Code	IN_4	IN_3	IN_2	IN_1	IN_0
0xXA	A	A	A	A	A
0xBB	D	A	A	A	A
0xCC	D	D	A	A	A
0xDD	D	D	D	A	A
0xEE	D	D	D	D	A
0xFF	D	D	D	D	D

X= Value from 0x0 to 0x3

A: analog input

D: digital input

Code	Vref+	Vref-
0x0X	Vdd	Vss
0x1X	IN_3	Vss
0x2X	Vdd	IN_2
0x3X	IN_3	IN_2

X= Value from 0xA to 0xF

Vref+/Vref-: reference input

Vdd: positive supply voltage

Vss: negative supply voltage

Returns

- 0 configuration entered
- 1 configuration code not allowed. Configuration not entered.

char configSerialPins(unsigned char pinConfigCode)

This function is used to configure pin 7 and 8 as serial pins or as digital input/output. At startup pin7/8 are configured as serial pins.

PinConfigCode =0 : Pin7= IN_5 ; Pin8= OUT_0;

pinConfigCode!=0 : Pin7= SER_RX ; Pin8= SER_TX

Returns

- 0 configuration performed.
- 1 pin7 IN_5/SER_RX configured wrongly as output
- 2 pin8 OUT_0/SER_TX configured wrongly as input

char readADIn(unsigned char inPin, unsigned char resolution, unsigned int *ADconversion)

Performs a measurement of the voltage at this input (inPin=0..4 for IN_0..4) using the built-in A/D converter with a resolution defined by the parameter resolution (in bit; max. 10 bit) and stores the result in *ADconversion.



This function may only be used with TCM220C hardware! TCM200C does not provide analog inputs!

Returns

- 0 A/D process performed
- 1 analog IN_0 not configured as analogue. A/D not performed
- 2 analog IN_1 not configured as analogue. A/D not performed
- 3 analog IN_2 not configured as analogue. A/D not performed
- 4 analog IN_3 not configured as analogue. A/D not performed
- 5 analog IN_4 not configured as analogue. A/D not performed
- 6 inPin value not allowed. A/D not performed
- 7 IN_0...IN_4 configured as output. A/D not performed

- 8 IN_0...IN_4 configured as output. A/D not performed
- 9 IN_0...IN_4 configured as output. A/D not performed
- 10 IN_0...IN_4 configured as output. A/D not performed
- 11 IN_0...IN_4 configured as output. A/D not performed
- 12 resolution value 0 or >10. A/D not performed
- 13 address pointed to by 2nd parameter not allowed. A/D not performed
- 14 IN_2 configured as Vref-. A/D not performed
- 15 IN_2 configured as Vref+. A/D not performed

char readDigitalIn(unsigned char inPin, unsigned char *level)

Reads the level of the input pin inPin (inPin =0 for IN_0 .. 5 for IN_5) and stores the result in *level (TRUE if HIGH level, FALSE if LOW level).

Returns

- 0 read process correct
- 1 IN_0 not configured as digital. Reading not performed (TCM220C only)
- 2 IN_1 not configured as digital. Reading not performed (TCM220C only)
- 3 IN_2 not configured as digital. Reading not performed (TCM220C only)
- 4 IN_3 not configured as digital. Reading not performed (TCM220C only)
- 5 IN_4 not configured as digital. Reading not performed (TCM220C only)
- 6 IN_5/SER_RX configured wrongly as SER_RX. Reading not performed
- 7 inPin value not allowed. Pin reading not performed
- 8 IN_0 configured as output. Reading not performed
- 9 IN_1 configured as output. Reading not performed
- 10 IN_2 configured as output. Reading not performed
- 11 IN_3 configured as output. Reading not performed
- 12 IN_4 configured as output. Reading not performed
- 13 IN_5/SER_RX configured as output. Reading not performed
- 14 Address pointed to by 2nd parameter not allowed . Reading not performed

unsigned long getTime()

This command returns a time stamp from the internal system clock. The time stamp is increased every ms (->restart from 0 after about 50 days). During sleep the timer is stopped.



During the transmission (also repeating!) and reception of radio telegrams the timer may be delayed by about 1 ms per telegram.

Returns

Current 32 bit time stamp [0x0-0xFFFFFFFF] in milliseconds

void nop()

Performs a non-instruction cycle. Each nop instruction lasts 100ns.

void wait(unsigned long delay)

Performs a delay of *delay* milliseconds. A clear watch dog timer is implemented to avoid a possible reset.

void clrWdt()

Clears the watch dog timer counter. Clear watch dog timer should be used preferably only once in the program. It should be written in a source code point which is executed on a regular basis. Not clearing WDT though this instruction within a WDT period provokes a reset of the μC . Only in sleep modus the program can stay longer than the WDT period.

void switchWdtOn()

Switches the WDT on. By default the WDT is on (`_CONFIG 3`).

void switchWdtOff()

Switches the WDT off. This allows saving around $6\mu\text{A}$ current consumption. If the WDT is off before going to sleep waking-up is only possible through a rising edge on pin 7 (`SER_RX`).

char writeFlash()

Writes the data in `flashBuffer[]` to FLASH. As the FLASH only provides a limited number of write cycles (typ. 1000) this function should be used with care.



It takes about 50ms to write to FLASH. If a watch dog timer period shorter than that is used the watch dog timer has to be switched off before execution of this command to avoid a reset!

```
extern unsigned char flashBuffer[256];
```

Returns

```
0    code correctly written in FLASH
1    Error during verify
```

Data should be modified as long as possible in RAM using the `flashBuffer[]`. Only if really needed the `flashBuffer[]` should be written to FLASH due to the limited number of write cycles. It is a good idea to create a pointer structure to organize the memory such as

```
struct memory{
long ID1;
long ID2;
char temperature;
...
}*MEMORY;
```

and to make `*MEMORY` point to the starting address of `flashBuffer[]`

```
MEMORY=(struct memory *)flashBuffer;
```

Now it is for example possible to access ID1 via `MEMORY->ID1`.

It must be made sure that the size of the memory structure does not exceed the size of the `flashBuffer[]`!

char readFlash()

Reads data from FLASH into the array flashBuffer[]; The execution of this function requires about 800µs.

```
extern unsigned char flashBuffer[256];
```

Returns

0 code correctly read from FLASH

unsigned char getNoiseLevel()

This function returns a number between 0 and 6 which gives an information on the background noise level. It can be used for analysis purposes like the RSSI level of the former RCM130C module.

Low 0 ... 6 high noise

A. Development Tools

A.1 Microchip Tools

Microchip offers a tool suite for the PIC 18F65J11 micro controller. EnOcean recommends to use the following tools:

- Microchip MPLAB® ICD 2 in-circuit debugger
- Microchip MPLAB® IDE, version 8.00
Please find version 8.00 on the CD-ROM. The license terms and conditions of Microchip apply! The IDE is also available for free download from www.microchip.com
- Microchip MPLAB® C18 C-Compiler, version 3.14. Please find the full featured 60 day Student Edition on the CD-ROM. The license terms and conditions of Microchip apply! The 60-day student edition is also available for free download from www.microchip.com.

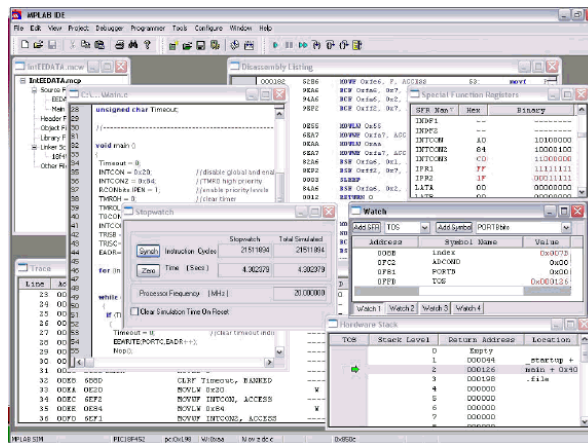
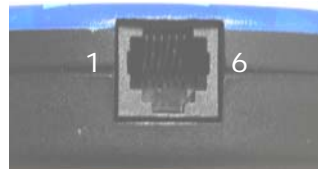


Figure 5: Microchip ICD 2 and MPLAB®

Pin-out of the ICD 2 connector:

Color	Description
1	Not used
2	Programming Clock
3	Programming Data
4	GND
5	Vdd
6	MCLR/Vpp



A.2 EnOcean Tools

EnOcean offers a development kit EDK100C containing – amongst other things – an evaluation board, an adapter for the MPLAB® ICD2 device from Microchip and 2 modified TCM200C modules¹ with mounted programming port connector. The evaluation board originally has been developed for use with RCM110/120 and TCM120. It is also possible to use it for the development of applications for TCM200C.



Figure 6: EnOcean EVA 110 board

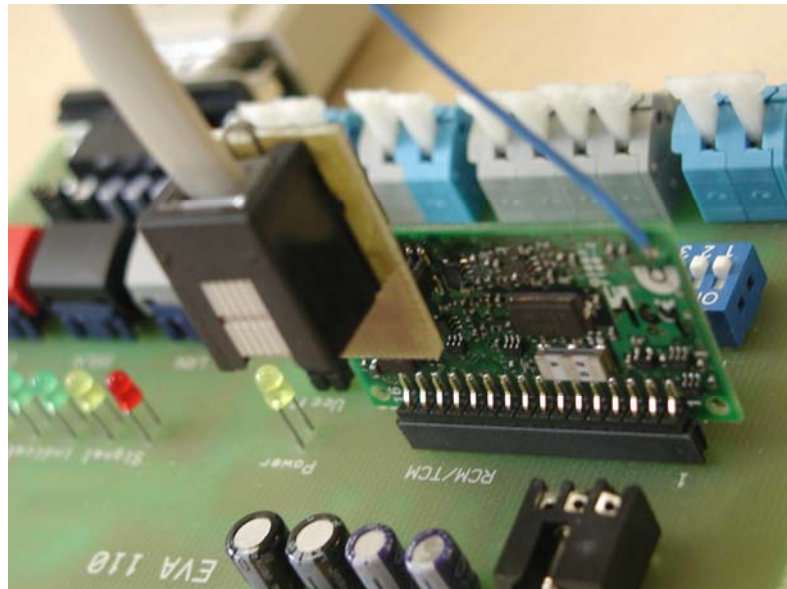


Figure 7: Adapter for MPLAB® ICD 2

¹ Important note: The expansion port connector, which is mounted for development purposes on the modules supplied with EDK100C is not mounted on standard TCM200C/TCM220C modules. In order to program the modules with own software in series production it is necessary to contact the programming pins with needles.

The meaning of connectors, indicators and switches are however different from the description in the EVA 100 manual.

Description of On-board Connectors

Symbol	Function	Operational characteristics
Adapter	Female jack for power supply	14 - 30 V, 100 mA max.
RCM/TCM	Female header for plug-in RCM or TCM module (Pin 1 is module antenna side)	
Test Connector	Female header connected directly to module leads.	
Vcc	Jack to disconnect the power supply to the inserted module	Bridge
GND GND	Ground connectors for functional control outputs	
R0 R1 R2 R3	Functional control outputs directly connected to the OUT_0..3 output pins of inserted module	Open drain or open collector outputs: 35 V max., 100 mA max., 100 mW max. each
GND EVG	Analog output to drive an Electronic Control Gear (Controlled via PWM on OUT_0, to be implemented in user application)	0 to 10 V, 20 mA max.
GND PWM	No function	
RS232	DB9 female serial interface connector to PC <ul style="list-style-type: none"> RCM 120, Operating Mode 0 (Rx) TCM 120 (Rx and Tx) TCM 200C (Rx and Tx) 	12 V
TCM RCM	Selector for RCM or TCM module operation	Bridge

Description of On-board Switches

Symbol	Function
MODE	Connected to IN_0...IN_2. Switching on connects the corresponding input to GND
LRN (Grey)	Connected to IN_3. Pressing the button connects input to GND.
SSLM (Black)	Connected to IN_4. Pressing the button connects input to GND.
CLR (Red)	Connected to IN_5. Pressing the button connects input to GND.

Description of On-board Indicators

Symbol	Function
Power	Power supply indicator
Signal indicator	No function
I0 I1 I2 I3	Status indicator for OUT_0 .. OUT_3 ("1" = LED on)
LMI	Status indicator for OUT_4 ("1" = LED on)

B. Installation

- Install the MPLAB® environment (IDE, C-Compiler and ICD 2)
- Unpack the API200_Vxxxx.zip in one folder
- Copy clib.lib, p18f65j11.lib and p18f65j11.h from the installation directory of your MPLAB® C18 C-compiler to this folder

After that you can start MPLAB® and open the project MyAPI200Project.mcp.

If everything is installed correctly you should now be able to compile the project using the "Build All" button.

C. Sample Applications

C.1 Default Application

This is the default application of TCM200C. The program sends radio telegrams received through the serial port, using the EnOcean serial protocol. The rx radio sensitivity can be set through the pin IN_3. The program allows additionally the activation of the repeater functionality through the pin IN_2 at start-up.

```

/*****
      Sw Description:      TCM200 main program
*****/

// *****/Configuration registers defines.*****/
//Watch dog timer period and brown-out detection configuration
#define _CONFIG          3 //watch dog timer 2048 ms
#define _NR_RX_RADIO_BUFFERS 4 //Nr of rx radio buffers. Value must be within interval [4,10]
#define _NR_TX_RADIO_BUFFERS 5 //Nr of tx radio buffers. Maximal value 11. With repeater value in interval
                               // [_NR_RX_RADIO_BUFFERS+1,11]
// *****/Configuration registers defines.*****/

// *****/Include files.*****/
#include "p18f65j11.h" //PIC names declaration file
#include "API200_LIB.h" //API200 api functions, variables and structure declaration file
#include "API200_CFG.h" //API200 configuration register file
// *****/Include files.*****/

void main(void){

    union telegram t1;
    unsigned char ucState,ucSens,result;

    result = initAPI200(); // TCM200 init function. This function must always be the first program line.
                          // It initializes the TCM200.
    ucSens = 1; // Radio rx sensitivity is high at startup

    t1.type = 0x0B; // RRT serial telegrams (they transmit a radio information)
                  // have this code
    result = receiverOn(); // Activating the rx radio.

    result = readDigitalIn(2,&ucState); // Reading the pin 4 (IN_2). The digital reading is stored in ucState.
    if (ucState == 0)
    {
        result = repeaterOn(); // Switching on the if the IN_2s is GND at startup
    }

    while(1) // Main loop
    {
        clrWdt(); // Clear the watchdog timer counter to avoid a reset

        if(!getRadioTelegram(&t1)) // Radio telegram received?
        {
            result = sendSerialTelegram(&t1); // Yes. Sending then the received info through the serial port.
        }

        result = readDigitalIn(3,&ucState); //Reading pin 5 (IN_3). The digital reading is stored in ucState.
        if (ucState != ucSens)
        {
            result = setRxSensitivity(ucState); // Setting a new radio sensitivity according to IN_3
            ucSens = ucState; // Updating the state of the pin for the next comparison
        }
    }
}

```

C.1 Function to show noise level with 3 LEDs

```
void showNoiseLevel()
{
static unsigned char ucNoiseLevel;

if (ucNoiseLevel==getNoiseLevel())
return;

ucNoiseLevel=getNoiseLevel();           //Here, the noise level has changed.

if (ucNoiseLevel>0x05)
    { writeOut(1,1); writeOut(2,1); writeOut(3,1);    }
else if (ucNoiseLevel>0x03)
    { writeOut(1,1); writeOut(2,1); writeOut(3,0);    }
else if (ucNoiseLevel>0x01)
    { writeOut(1,1); writeOut(2,0); writeOut(3,0);    }
else
    { writeOut(1,0); writeOut(2,0); writeOut(3,0);    }
}
```