

IQRF OS

Operating System

Version 3.00

Reference Guide



Simple way to smarter wireless solutions

Quick reference

Values between system functions and superordinate program are passed on via parameters. OS uses 4 parameters in total: param1 (1 B), param2 (1 B), param3 (2 B) and param4 (2 B). Their location in memory see the RAM map [2]. Individual functions have up to 3 parameters. Several functions use some of these params and W (PIC accumulator) to return output values. Note that they are valid until another function using the same parameter or the `debug` function is called by the user. Additionally, some functions use some params as work variables that is why their previous content can be destroyed.

Only two stack levels are available to call OS functions in subroutines. Exceptions see `RFTXpacket`, `RFRXpacket`, `answerSystempacket`, `bondRequest` and `bondNewNode`.

Functions

Control		4
<code>reset()</code>	Initialization of microcontroller, OS and application	4
<code>calibrateTimer()</code>	Calibrate tick generator	4
<code>iqrfsleep()</code>	Set the TR module in power saving mode (Sleep)	5
<code>setRFsleep()</code>	Set the RF IC in power saving mode (Sleep)	5
<code>setRFready()</code>	Set the RF IC in ready mode (wake-up from Sleep)	6
<code>debug()</code>	Enter the debug mode	6
<code>uns8 getSupplyVoltage()</code>	Get voltage level for battery check	7
<code>getTemperature()</code>	Temperature measurement	7
Active waiting		8
<code>waitMS(ms)</code>	Active waiting (time in ms)	8
<code>waitDelay(ticks)</code>	Active waiting (time in ticks)	8
<code>waitNewTickDelay()</code>	Wait for a new tick	9
Timing on background		9
<code>startDelay(ticks)</code>	Start waiting (time in ticks)	10
<code>startLongDelay(ticks)</code>	Start long waiting (time in ticks)	11
<code>bit isDelay()</code>	Still waiting	11
<code>startCapture()</code>	Resets counter of ticks	9
<code>captureTicks()</code>	Get number of ticks counted	10
LED indication		12
<code>setOnPulsingLED(ticks)</code>	LEDR and LEDG On times setting (for blinking)	12
<code>setOffPulsingLED(ticks)</code>	LEDR and LEDG Off times setting (for blinking)	12
<code>pulsingLEDR()</code>	Red LED activation (blinking on background)	13
<code>pulseLEDR()</code>	Single red LED pulse (one flash on background)	13
<code>stopLEDR()</code>	Red LED off, blinking stopped	14
<code>pulsingLEDG()</code>	Green LED activation (blinking on background)	14
<code>pulseLEDG()</code>	Single green LED pulse (one flash on background)	15
<code>stopLEDG()</code>	Green LED off, blinking stopped	15
EEPROM		16
<code>uns8 eeReadByte(addr)</code>	Read one byte	16
<code>eeReadData(addr, length)</code>	Read a block	16
<code>eeWriteByte(addr, data)</code>	Write one byte	17
<code>eeWriteData(addr, length)</code>	Write a block	17
RAM		18
<code>uns8 readFromRAM(addr)</code>	Read one byte	18
<code>writeToRAM(addr, data)</code>	Write one byte	19
Buffers, data blocks		20
<code>clearBufferINFO()</code>	bufferINFO clearing	24
<code>clearBufferRF()</code>	bufferRF clearing	24
<code>copyBufferINFO2COM()</code>	Copy bufferINFO to bufferCOM	20
<code>copyBufferINFO2RF()</code>	Copy bufferINFO to bufferRF	20
<code>copyBufferRF2COM()</code>	Copy bufferRF to bufferCOM	21
<code>copyBufferRF2INFO()</code>	Copy bufferRF to bufferINFO	21
<code>copyBufferCOM2RF()</code>	Copy bufferCOM to bufferRF	21
<code>copyBufferCOM2INFO()</code>	Copy bufferCOM to bufferINFO	22
<code>bit compareBufferINFO2RF(length)</code>	Comparison of bufferINFO and bufferRF	22
<code>void swapBufferINFO()</code>	Swap bufferINFO and bufferAUX	23
<code>copyMemoryBlock(uns16 from, uns16 to, uns8 length)</code>	Copy any data block to any position	23

<code>moduleInfo()</code>	Get info about transceiver module and OS	25
<code>appInfo()</code>	Copy info about application from EEPROM to <code>bufferINFO</code>	25
SPI		26
<code>enableSPI()</code>	SPI communication line activation	26
<code>disableSPI()</code>	SPI communication line deactivation	26
<code>startSPI(length)</code>	SPI packet transmission	27
<code>stopSPI()</code>	SPI stopping	28
<code>restartSPI()</code>	SPI continuing	28
<code>bit getStatusSPI()</code>	SPI status, update SPI flags	29
RF		30
<code>setTXpower(level)</code>	RF power setting (7 levels)	30
<code>setRFspeed(speed)</code>	Select RF bit rate	30
<code>setRFband(band)</code>	Select RF band (868 MHz or 916 MHz)	31
<code>setRFchannel(channel)</code>	Select RF channel	31
<code>setRFmode(mode)</code>	Select RF power management mode	32
<code>checkRF(level)</code>	Detect incoming RF signal	33
<code>RFTXpacket()</code>	Send a packet from <code>bufferRF</code> via RF	36
<code>bit RFRXpacket()</code>	Receive a packet via RF to <code>bufferRF</code>	36
Networking		37
<code>setCoordinatorMode()</code>	Device is the Coordinator	37
<code>setNodeMode()</code>	Device is a Node	37
<code>setNonetMode()</code>	Networking disabled	38
<code>setNetworkFilteringOn()</code>	Packets accepted from current network only	39
<code>setNetworkFilteringOff()</code>	Packets accepted from both networks	39
<code>setUserAddress(uns16: address)</code>	Assign a user address to a Node	40
<code>uns8 getNetworkParams()</code>	Get information about the network	41
Routing		42
<code>setRoutingOn()</code>	Outgoing packets routed via other devices on background	42
<code>setRoutingOff()</code>	No routing for outgoing packets	42
<code>uns8 discovery(zones)</code>	Discover Nodes for routing	43
<code>answerSystemPacket()</code>	Enable response to Coordinator for Discovery	44
<code>bit isDiscoveredNode(N)</code>	Check for being discovered	45
<code>bit wasRouted()</code>	Indicate incoming packet routing	45
<code>optimizeHops(x)</code>	Optimize number of hops for given Node	46
Bonding - Node		47
<code>bit bondRequest()</code>	Request for bonding	47
<code>bit amIBonded()</code>	Is the Node bonded?	48
<code>removeBond()</code>	Unbonding	48
Bonding - Coordinator		49
<code>bit bondNewNode(address)</code>	Bonding a Node	49
<code>bit isBondedNode(N)</code>	Is the Node bonded?	50
<code>removeBondedNode(N)</code>	Unbonding a Node	50
<code>bit rebondNode(N)</code>	Rebonding a Node	51
<code>clearAllBonds()</code>	Clearing of all bonds	51

Unless otherwise stated, all functions are the `void` type and all their parameters are the `uns8` type.

OS functions

Control

reset

Function	Initialization of microcontroller, OS and application
Purpose	If needed, it is possible to initialize the application and run the program from the very beginning again.
Syntax	<code>void reset ()</code>
Parameters	–
Return value	–
Output values	<ul style="list-style-type: none"> All user RAM is cleared. The only register which keeps the original us value is <code>userStatus</code> – see the IQRF OS User's guide. <code>-TO = 0</code>
Preconditions	–
Remarks	<ul style="list-style-type: none"> This is forced watchdog overflow reset. To identify this type of reset the <code>-TO</code> and other status flags are available in the <code>userReg0</code> just after boot. See IQRF OS User's guide [1] (Reset). Another helping way to analyze reset causation is via the <code>userStatus</code> register.
Side effects	It is strictly specified which RAM registers are initialized and which are unchanged. Refer to the PIC datasheet [8] (Reset) for details.
See also	–
Example	<pre>if (Error == 1) reset();</pre>

calibrateTimer

Function	Calibrate tick generator
Purpose	Precise timing
Syntax	<code>void calibrateTimer ()</code>
Parameters	–
Return value	–
Output values	Tick duration is calibrated to eliminate RC oscillator inaccuracy.
Preconditions	–
Remarks	<ul style="list-style-type: none"> Calibration takes ~20 ms It is automatically done after reset. Recommended from time to time, after rapid temperature or supply voltage change etc. See PIC datasheet, Oscillator parameters, frequency vs. V_{DD} and Temperature.
Side effects	–
See also	–
Example	<pre>if ((NextHour = 1) or (TempChange > 5)) // Calibrate once an hour calibrateTimer(); // or after rapid temperature change</pre>

iqrfSleep

Function	Setting the TR module in power saving mode (Sleep)
Purpose	Easy and efficient power management. This function, once called, puts the module into the Sleep mode. Wake-up can be caused by power off/on, watchdog timeout or on the C5 pin change.
Syntax	<code>void iqrfSleep ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	This functions operates like the PIC machine instruction Sleep. Additionally, OS suspends all HW resources that are under its control (RF circuitry, timers, internal PIC pins, LEDs etc.). The user should do the same for resources used by the application before entering the Sleep mode to achieve minimal power consumption. For wake-up on pin change the required sequence should be executed. Wake-up on pin change is default disabled.
Remarks	All features are under user's control. RBIF flag is not cleared to allow to distinguish wake-up type. See example E01-TX [10].
Side effects	Global interrupt enable (GIE) is controlled by OS again after wake-up.
See also	setRFsleep
Example1	<pre> // minimize consumption (depends on resources used by the user) Motor = 0; // Stop the motor ADON = 0; // Disable A/D converter SWDTEN = 0; // Disable watchdog iqrfSleep(); // Put the module into Sleep mode </pre>
Example2	<pre> // wake-up on pin change. See example E01-TX. GIE = 0; // Disable all interrupts RBIE = 1; // Enable wake-up on pin change SWDTEN = 0; // Disable watchdog to save ~2µA iqrfSleep(); // Put the module into Sleep mode RBIF = 0; // Clear flag if (buttonPressed) // If button is pressed { ... } // ... </pre>

setRFsleep

Function	Setting RF circuitry in power saving mode (Sleep)
Purpose	To put all RF circuitry in Sleep mode. Easy and efficient power management.
Syntax	<code>void setRFsleep ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	0.6 mA typ. is saved. RF response is prolonged for 2 ms typ., 7 ms max. due to wake-up. Wake-up can be caused by setRFready, RFTXpacket, RFRXpacket, checkRF or getSupplyVoltage.
Side effects	–
See also	setRFready, iqrfSleep, getSupplyVoltage, checkRF, RFTXpacket, RFRXpacket
Example	<code>setRFsleep(); // Put the RF circuitry in Sleep mode</code>

setRFready

Function	Wake RF circuitry up
Purpose	To wake RF circuitry up in advance for faster response. Easy and efficient power management.
Syntax	<code>void setRFready ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	–
Side effects	–
See also	<code>setRFsleep</code> , <code>iqrfsleep</code> , <code>getSupplyVoltage</code> , <code>checkRF</code> , <code>RFTXpacket</code> , <code>RFRXpacket</code>
Example	<pre>setRFready(); // Wake the RF circuitry up from RF sleep in advance ... // at least 2 ms before RF operation RFTXpacket(); // for immediate reaction</pre>

debug

Function	Enter the debug mode
Purpose	IQRF OS directly supports debugging and testing. It is possible to stop the application wherever you need and display internal values (variables, RAM registers, EEPROM etc.) and then continue later on.
Syntax	<code>void debug ()</code>
Parameters	–
Return value	–
Output values	OS directly returns no value but supports using W (PIC accumulator) to identify which of the debug points is currently active.
Preconditions	<ul style="list-style-type: none"> • Debug should be used with corresponding development kit (e.g. CK-USB-04) and the IQRF IDE development environment. • To avoid possible HW collision with respect to user application, <code>debug</code> operates only under the following conditions: <ul style="list-style-type: none"> • Pins C5 to C8 are configured for SPI slave in respective TRIS bits (C8 out, the others in). It is arranged by OS by default. • The <i>Check Mode</i> function is enabled in IQRF IDE. Otherwise no communication on these pins is initiated by debug tools even though TR is in debug mode until the <i>Check Mode</i> is enabled. • SPI need not be enabled by <code>enableSPI</code>
Remarks	Number of <code>debug ()</code> instances is unlimited. The application is running until a <code>debug</code> function is encountered. Then the program is stopped and the module is switched to the debug mode allowing IQRF IDE to display values. The module stays in the debug mode until the user selects the <i>Skip Debug</i> button. Then the application program continues running until another <code>debug</code> function is encountered and so on. See IQRF IDE Help and example E04-EEPROM [10].
Side effects	<ul style="list-style-type: none"> • <code>param1</code> to <code>param4</code> are not displayed • Watchdog is cleared while in Debug mode
See also	–
Example	<pre>if (compareBufferINFO2RF(4)) W = 1; // match else W = 2; // mismatch debug(); // Skip Debug 1 or 2 will be displayed here according the result</pre>

getSupplyVoltage

Function	Power supply measurement (up to 3.8 V)
Purpose	Battery check for discharge-sensitive batteries
Syntax	uns8 getSupplyVoltage ()
Parameters	–
Return value	level = 1, 2, ...15 Voltage > 2.25 V + level × 0.1 V
Output values	–
Preconditions	–
Remarks	<ul style="list-style-type: none"> • Internal power supply voltage is checked. • In case of TR-52B it is the LDO output but not actual battery voltage. This value is 3.0 V typ. if battery is O.K. and drops down if battery is low. • To evaluate the battery, take into consideration your battery type and power supply circuitry with respect to diodes and other possible voltage drops. • The detector circuit has a built-in 50 mV hysteresis.
Side effects	The RF circuitry wakes up (in case of sleeping).
See also	setRFsleep
Example	<pre>if (getSupplyVoltage () > 7) ... // Voltage > 2.95V else ... // Low battery</pre>

getTemperature

Function	For TR-52B only: Converts the analog temperature sensor output to digital value.
Purpose	Temperature measurement
Syntax	void getTemperature ()
Parameters	–
Return value	–
Output values	Unsigned 10b result is stored in ADRES (ADRESH and ADRESL PIC special function registers), right justified.
Preconditions	–
Remarks	Temperature can be measured with on-board temperature sensor and internal A/D converter of the microcontroller. Temperature (in °C) evaluation: $T_a = ADRES \times 75/256 - 50$. See example E08–TEMPERATURE [10].
Side effects	–
See also	–
Example1	<pre>// Temperature measurement uns16 temperature; getTemperature (); // Temperature measurement temperature.high8 = ADRESH&0x03; // 10 b result is stored in ADRESH temperature.low8 = ADRESL; // and ADRESL</pre>
Example2	<pre>// Conversion to °C temperature = 75/256 * temperature - 50 [°C] temperature *= 75; temperature >>= 8; temperature -= 50; // temperature.high8 = tens of °C // temperature.high8 = units of °C</pre>

Active waiting

waitMS

Function	Wait specified number of miliseconds
Purpose	Time delay generation
Syntax	<code>void waitMS (ms)</code>
Parameters	ms - time to wait in miliseconds (1 - 255)
Return value	–
Output values	–
Preconditions	This function can be combined with <code>waitDelay</code> , <code>startCapture</code> and <code>captureTicks</code> .
Remarks	This is an active waiting (on OS foreground). No other operation runs on OS foreground during waiting. Time precision depends on internal RC oscillator. Thus, the delay can vary with temperature etc. See respective PIC datasheet [8].
Side effects	–
See also	<code>waitDelay</code> , <code>startDelay</code> , <code>startLongDelay</code>
Example	<pre>waitMS(10); // Delay 10 ms. Program stays here for the whole 10 ms period ... // and continues here just after the period elapsed.</pre>

waitDelay

Function	Wait specified number of ticks
Purpose	Time delay generation
Syntax	<code>void waitDelay (ticks)</code>
Parameters	ticks – time to wait in 10 ms intervals (1 - 255)
Return value	–
Output values	–
Preconditions	This function can be combined with <code>waitMS</code> .
Remarks	This is the active waiting (on OS foreground). No other operation runs on OS foreground during waiting.
Side effects	This function should not be combined with <code>startDelay</code> and <code>startLongDelay</code> . Internal ticks are based on internal RC oscillator. Thus, the delay can vary with temperature etc. See respective PIC datasheet [8]. For short time delays <code>waitMS</code> is more precise due to latency.
See also	<code>waitMS</code> , <code>startDelay</code> , <code>startLongDelay</code>
Example	<pre> // LED on for 0.5 s _LED = 1; waitDelay(50); // Delay 500 ms. Program stays here for 500 ms _LED = 0; // and continues here just after the period elapsed.</pre>

waitNewTick

Function	Wait for a new tick
Purpose	Timing synchronization of user operations
Syntax	<code>void waitNewTick ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	Active waiting (on OS foreground) until a new tick starts. No other operation runs on OS foreground during this waiting.
Side effects	–
See also	<code>waitMS</code> , <code>waitDelay</code>
Example	<code>waitNewTick ();</code>

Timing on background

startCapture

Function	Reset and start the Capture timer
Purpose	Initialization of time measurement or delay generation
Syntax	<code>void startCapture ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	This function can be combined with <code>waitMS</code> .
Remarks	Capture timer is a resettable counter of OS ticks (10 ms system intervals) running on OS background. This function clears the counter and starts counting.
Side effects	Functionality is affected by <code>bondRequest</code> , <code>bondNewNode</code> and <code>RFRXpacket</code> .
See also	<code>captureTicks</code>
Example	See <code>captureTicks</code>

captureTicks

Function	Get number of ticks counted from the last <code>startCapture</code> and <code>captureTicks</code> calling.
Purpose	Measurement of elapsed time.
Syntax	<code>void captureTicks ()</code>
Parameters	–
Return value	–
Output value	<ul style="list-style-type: none"> • param3: ticks counted from the last <code>startCapture</code> (0 - 65535) • param4: ticks counted from the last <code>captureTicks</code> (0 - 65535)
Preconditions	<ul style="list-style-type: none"> • <code>startCapture</code> should be used at least once before. • To ensure correct operation the counter must not overflow. That is why <code>captureTicks</code> should be called max. ~655 s after last <code>startCapture</code> or <code>captureTicks</code> calling.
Remarks	See example E05–DELAYS [10]
Side effects	Functionality is affected by <code>bondRequest</code> , <code>bondNewNode</code> and <code>RFRXpacket</code> . Internal ticks are based on internal RC oscillator. Thus, the delay can vary with temperature etc. See respective PIC datasheet [8].
See also	<code>startCapture</code>
Example	<pre>startCapture(); // Reset counter of ticks waitMS(200); // Delay 200 ms captureTicks(); // param3 == 20 waitMS(150); // Delay 150 ms captureTicks(); // param3 == 35, param4 == 15 startCapture(); // Reset counter of ticks waitMS(100); // Delay 100 ms captureTicks(); // param3 == 10</pre>

startDelay

Function	Preset and start the Delay timer
Purpose	Initialization of time measurement or delay generation
Syntax	<code>void startDelay (ticks)</code>
Parameters	<code>uns8 ticks</code> : number of ticks (10 ms system intervals) to be measured (1-255)
Return value	–
Output values	–
Preconditions	This function can be combined with <code>waitMS</code> and <code>startLongDelay</code> .
Remarks	The Delay timer measures specified time period on OS background. The result is available via the <code>isDelay</code> function.
Side effects	This function does not work properly if the <code>waitDelay</code> , <code>startLongDelay</code> functions are active.
See also	<code>isDelay</code> , <code>startLongDelay</code> , <code>waitDelay</code>
Example	See <code>isDelay</code>

startLongDelay

Function	Preset and start the LongDelay timer
Purpose	Initialization of time measurement or delay generation
Syntax	void startLongDelay (ticks)
Parameters	uns16 ticks: number of ticks (10 ms system intervals) to be measured (1-65535)
Return value	–
Output values	–
Preconditions	This function can be combined with <code>waitMS</code> and <code>startDelay</code> .
Remarks	The Delay timer measures specified time period on OS background. The result is available via the <code>isDelay</code> function.
Side effects	This function does not work properly if the <code>waitDelay</code> , <code>startDelay</code> functions are active.
See also	<code>isDelay</code> , <code>startDelay</code> , <code>waitDelay</code>
Example	See <code>isDelay</code>

isDelay

Function	Information whether specified delay is still in progress
Purpose	Time measurement or delay generation
Syntax	bit isDelay ()
Parameters	–
Return value	<ul style="list-style-type: none"> • 1: still in progress • 0: elapsed
Output values	–
Preconditions	<code>startDelay</code> or <code>startLongDelay</code> should be used before.
Remarks	<ul style="list-style-type: none"> • The (Long)Delay timer measures specified time period. The result is available via the <code>isDelay</code> function. • Tip: the <code>clrwdt</code> instruction should be used to avoid unintentional watchdog reset during the delay. • See example E05-DELAYS [10].
Side effects	–
See also	<code>startDelay</code> , <code>startLongDelay</code>
Example1	<pre> // LED on for 1 s _LED = 1; startDelay(100); // Start 1 sec delay counting on OS background while (isDelay()) // Wait until the delay is over { clrwdt(); // Any useful operation on OS foreground can be ... // performed during waiting } _LED = 0; // Continue here after 1 sec </pre>
Example2	<pre> // LED on for 10 s _LED = 1; startLongDelay(1000); // Start 10 sec delay counting on OS background while (isDelay()) // Wait until the delay is over { clrwdt(); // Any useful operation on OS foreground can be ... // performed during waiting } _LED = 0; // Continue here after 10 sec </pre>

LED indication

setOnPulsingLED

Function	LEDs On time setting (red as well as green)
Purpose	Specification of the "On" time for LEDs (either for a single flash or for blinking)
Syntax	<code>void setOnPulsingLED (ticks)</code>
Parameters	<code>uns8 ticks</code> : number of ticks (10 ms system intervals) (1-255)
Return value	–
Output values	–
Preconditions	–
Remarks	Default value is 5 (50 ms).
Side effects	–
See also	<code>setOffPulsingLED</code> , <code>pulsingLEDR</code> , <code>pulseLEDR</code> , <code>pulsingLEDG</code> , <code>pulseLEDG</code>
Example	See <code>setOffPulsingLED</code>

setOffPulsingLED

Function	LEDs Off time setting (red as well as green)
Purpose	Specification of the "Off" time for LEDs (for blinking)
Syntax	<code>void setOffPulsingLED (ticks)</code>
Parameters	<code>uns8 ticks</code> : number of ticks (10 ms system intervals) (1-255)
Return value	–
Output values	–
Preconditions	–
Remarks	Default value is 20 (200 ms).
Side effects	–
See also	<code>setOnPulsingLED</code> , <code>pulsingLEDR</code> , <code>pulsingLEDG</code>
Example	<pre> // Change blinking to 250 ms On / 750 ms Off setOnPulsingLED(25); // 250 ms On setOffPulsingLED(75); // 750 ms Off </pre>

pulsingLEDR

Function	Red LED blinking
Purpose	Continuous red LED blinking on OS background
Syntax	<code>void pulsingLEDR ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Blinking times should be defined in advance by <code>setOnPulsingLED</code> and <code>setOffPulsingLED</code> . The appropriate PIC pin is configured as an output automatically.
Remarks	Blinking continues until it is stopped by the user (e.g. by <code>stopLEDR</code>).
Side effects	–
See also	<code>setOnPulsingLED</code> , <code>setOffPulsingLED</code> , <code>stopLEDR</code> , <code>pulseLEDR</code>
Example1	<code>pulsingLEDR (); // continuous blinking on OS background</code>
Example1	<pre>// Blinking for 2 s pulsingLEDR (); // blinking for 2 s on OS background waitDelay(200); // 2 s delay generated on foreground stopLEDR (); // Stop blinking</pre>

pulseLEDR

Function	Single red LED flash
Purpose	Red LED flash on OS background
Syntax	<code>void pulseLEDR ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Flash time should be defined in advance by <code>setOnPulsingLED</code> . The appropriate PIC pin is configured as an output automatically.
Remarks	The on-board LEDs can also be directly controlled on OS foreground using C commands for manipulating the <code>_LEDR</code> output (the pin the red LED is connected to) and corresponding control bit (<code>TRISx.x</code> - see <code>IQRF-memory.h</code> header file).
Side effects	<ul style="list-style-type: none"> • The appropriate PIC pin is not restored to the state before <code>pulsingLEDR/pulseLEDR</code> (<code>TRISx.x == 0, _LEDR == 0</code> after finishing on background). • Background activity overrides setting of <code>_LEDR</code> (the on-board red LED pin) made by the application. Thus, attention should be taken when using direct setup of LED output pin combined with LED functions. Possible background LED routines can overwrite the status set by direct manipulation with the pin.
See also	<code>setOnPulsingLEDR</code> , <code>pulsingLEDR</code> , <code>stopLEDR</code>
Example	<pre>setOnPulsingLEDR(10); // 100 ms On pulseLEDR (); // Single red LED flash for 100 ms on OS background ... // Program continues immediately, // not waiting until the delay expires. // LED will be switched off after 100 ms automatically</pre>

stopLEDR

Function	Red LED off, blinking stopped
Purpose	Stops the red LED activity on OS background
Syntax	<code>void stopLEDR ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	–
Side effects	<ul style="list-style-type: none"> • The appropriate PIC pin is not restored to the state before pulsingLEDR/pulseLEDR (TRISx.x == 0, _LEDR == 0 after finishing on background). • Background activity overrides setting of _LEDR (the on-board LED pin) made by the application. Thus, attention should be taken when using direct setup of LED output pin combined with LED functions. Possible background LED routines can overwrite the status set by direct manipulation with the pin.
See also	pulsingLEDR, pulseLEDR
Example1	<pre>pulsingLEDR(); // Start blinking on OS background ... // Blinking continues during any operation stopLEDR(); // Stop blinking</pre>
Example2	<pre>pulseLEDR(); // Red LED On on OS background ... // continuously lighting during any operation // until specified time expired stopLEDR(); // or LED is switched Off by this command</pre>

pulsingLEDG

Function	Green LED blinking
Purpose	Continuous green LED blinking on OS background
Syntax	<code>void pulsingLEDG ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Blinking times should be defined in advance by setOnPulsingLED and setOffPulsingLED. The appropriate PIC pin is configured as an output automatically.
Remarks	Blinking continues until it is stopped by the user (e.g. by stopLEDG).
Side effects	–
See also	setOnPulsingLED, setOffPulsingLED, stopLEDG, pulseLEDG
Example1	<pre>pulsingLEDG(); // continuous blinking on OS background</pre>
Example1	<pre>// Blinking for 2 s pulsingLEDG(); // blinking for 2 s on OS background waitDelay(200); // 2 s delay generated on foreground stopLEDG(); // Stop blinking</pre>

pulseLEDG

Function	Single green LED flash
Purpose	Green LED flash on OS background
Syntax	<code>void pulseLEDG ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Flash time should be defined in advance by <code>setOnPulsingLEDG</code> . The appropriate PIC pin is configured as an output automatically.
Remarks	The on-board LEDs can also be directly controlled on OS foreground using C commands for manipulating the <code>_LEDG</code> output (the pin the green LED is connected to) and corresponding control bit (<code>TRISx.x</code> - see <code>IQRF-memory.h</code> header file).
Side effects	<ul style="list-style-type: none"> • The appropriate PIC pin is not restored to the state before <code>pulsingLEDG/pulseLEDG</code> (<code>TRISx.x == 0, _LEDG == 0</code> after finishing on background). • Background <code>activity</code> overrides setting of <code>_LEDG</code> (the on-board green LED pin) made by the application. Thus, attention should be taken when using direct setup of LED output pin combined with LED functions. Possible background LED routines can overwrite the status set by direct manipulation with the pin.
See also	<code>setOnPulsingLEDG, pulsingLEDG, stopLEDG</code>
Example	<pre>setOnPulsingLEDG(10); // 100 ms On pulseLEDG(); // Single green LED flash for 100 ms on OS background ... // Program continues immediately, // not waiting until the delay expires. // LED will be switched off after 100 ms automatically</pre>

stopLEDG

Function	Green LED off, blinking stopped
Purpose	Stops the green LED activity on OS background
Syntax	<code>void stopLEDG ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	–
Side effects	<ul style="list-style-type: none"> • The appropriate PIC pin is not restored to the state before <code>pulsingLEDG/pulseLEDG</code> (<code>TRISx.x == 0, _LEDG == 0</code> after finishing on background). • Background <code>activity</code> overrides setting of <code>_LEDG</code> (the on-board LED pin) made by the application. Thus, attention should be taken when using direct setup of LED output pin combined with LED functions. Possible background LED routines can overwrite the status set by direct manipulation with the pin.
See also	<code>pulsingLEDG, pulseLEDG</code>
Example1	<pre>pulsingLEDG(); // Start blinking on OS background ... // Blinking continues during any operation stopLEDG(); // Stop blinking</pre>
Example2	<pre>pulseLEDG(); // Green LED On on OS background ... // continuously lighting during any operation // until specified time expired // or LED is switched Off by this command stopLEDG();</pre>

EEPROM

eeReadByte

Function	Read one byte from specified location in EEPROM
Purpose	Access to EEPROM
Syntax	<code>uns8 eeReadByte (addr)</code>
Parameters	<code>uns8 addr</code> : address in EEPROM (0 to 0xBF). See EEPROM map [2].
Return value	Value read from specified EEPROM location (0 to 255)
Output values	–
Preconditions	–
Remarks	<ul style="list-style-type: none"> • Direct user access to EEPROM (using registers <code>EECONx</code> etc.) is not allowed for security reasons, specialized OS functions are intended for this. • EEPROM area dedicated to OS (locations 0xC0 or higher) is not accessible. See example E04–EEPROM [10].
Side effects	Any attempt to read from protected area above 0xBF leads to reading from EEPROM location 0xA0.
See also	<code>eeReadData</code> , <code>eeWriteByte</code> , <code>eeWriteData</code>
Example1	<pre>i = eeReadByte(0); // store 1 byte from EEPROM from address 0 to i</pre>
Example2	<pre>// Illegal access: Avoid access to EEPROM locations 192 (0xC0) or higher i = eeReadByte(200); // Reading from protected area is redirected to 160 (0xA0)</pre>

eeReadData

Function	Read a block of specified length from specified location in EEPROM to <code>bufferINFO</code>
Purpose	Block access to EEPROM
Syntax	<code>void eeReadData (addr, length)</code>
Parameters	<ul style="list-style-type: none"> • <code>uns8 addr</code>: address in EEPROM (0 to 0xBF - length + 1). See EEPROM map [2]. • <code>uns8 length</code>: number of bytes to be read (1 to 32)
Return value	–
Output values	<code>bufferINFO[0 to length - 1]</code>
Preconditions	–
Remarks	<ul style="list-style-type: none"> • Direct user access to EEPROM (using registers <code>EECONx</code> etc.) is not allowed for security reasons, specialized OS functions are intended for this. • EEPROM area dedicated to OS (locations 0xC0 or higher) is not accessible. See example E04–EEPROM [10].
Side effects	• Any attempt to read from protected area above 0xBF leads to reading from EEPROM location 0xA0.
See also	<code>eeReadByte</code> , <code>eeWriteByte</code> , <code>eeWriteData</code>
Example1	<pre>eeReadData(10, 16); // copy 16B from EEPROM from address 10 to bufferINFO // bufferINFO[0] = EEPROM[10] // ... // bufferINFO[15] = EEPROM[25]</pre>
Example2	<pre>// Illegal access: Avoid access to EEPROM locations 192 (0xC0) or higher eeReadData(200, 16); // EEPROM address 160 is used instead of protected area // bufferINFO[0] = EEPROM[160] // ... // bufferINFO[15] = EEPROM[160]</pre>

eeWriteByte

Function	Write one byte to specified location in EEPROM
Purpose	Access to EEPROM
Syntax	<code>void eeWriteByte (addr, data)</code>
Parameters	<ul style="list-style-type: none"> • <code>uns8 addr</code>: address in EEPROM (0xA0 to 0xBF for Coordinator and 0 to 0xBF for other devices). See EEPROM map [2]. • <code>uns8 data</code>: value to be written (0 to 255)
Return value	–
Output values	–
Preconditions	–
Remarks	<ul style="list-style-type: none"> • Direct user access to EEPROM (using registers <code>EECONx</code> etc.) is not allowed for security reasons, specialized OS functions are intended for this. • EEPROM area dedicated to OS (locations 0xC0 or higher) is not accessible. See example E04–EEPROM [10]. • Any attempt to write to protected area above 0xBF leads to no operation.
Side effects	–
See also	<code>eeReadByte</code> , <code>eeReadData</code> , <code>eeWriteData</code>
Example1	<pre>eeWriteByte(191, 0x75) // store 0x75 to EEPROM to address 191 eeWriteByte(0x80, X) // copy X to EEPROM to address 0x80</pre>
Example2	<pre>// Illegal access: Avoid access to EEPROM locations 192 (0xC0) or higher eeWriteByte(198, 0x75); // Attempt to write to protected area - nothing is written.</pre>

eeWriteData

Function	Write a block of specified length from <code>bufferINFO</code> to specified location in EEPROM
Purpose	Block access to EEPROM
Syntax	<code>void eeWriteData (addr, length)</code>
Parameters	<ul style="list-style-type: none"> • <code>uns8 addr</code>: address in EEPROM . See EEPROM map [2]. <ul style="list-style-type: none"> • (0xA0 to 0xBF - length + 1) for Coordinator • (0 to 0xBF - length + 1) for other devices • <code>uns8 length</code>: number of bytes to be written from <code>bufferINFO</code> (1 to 32)
Return value	–
Output values	–
Preconditions	–
Remarks	<ul style="list-style-type: none"> • Direct user access to EEPROM (using registers <code>EECONx</code> etc.) is not allowed for security reasons, specialized OS functions are intended for this. • EEPROM area dedicated to OS (locations 0xC0 or higher) is not accessible. See example E04–EEPROM [10].
Side effects	• Any attempt to write to protected area above 0xBF leads to no operation.
See also	<code>eeReadByte</code> , <code>eeReadData</code> , <code>eeWriteByte</code>
Example1	<pre>eeWriteData(10,16); // copy 16B from bufferINFO to EEPROM to address 10 // EEPROM[10] = bufferINFO[0] // // EEPROM[25] = bufferINFO[15]</pre>
Example2	<pre>// Illegal access: Avoid access to EEPROM locations 192 (0xC0) or higher eeWriteData(200,16); // Attempt to write to protected area - nothing is // written.</pre>

RAM
readFromRAM

Function	Read one byte from specified location in RAM
Purpose	Indirect access to RAM registers
Syntax	<code>uns8 readFromRAM(addr)</code>
Parameters	<code>uns8 addr</code> : memory location address. "Short" address within respective bank pair (up to 0xFF) should be used (e.g. 0xC5 for both 0xC5 and 0x1C5). <i>But using long addresses (up to 0x1FF, e.g. 0x1C5) also works because the compiler trims the overlapping bit to correct 8b value.</i> See Preconditions as well.
Return value	Value read from specified location
Output values	–
Preconditions	Before using this function, the IRP bank select bit must correspond to the location of the register (IRP=0 for RAM bank 0 and 1, IRP=1 for RAM bank 2 and 3). See the PIC datasheet [8] (Memory organization) and RAM map [2] for details.
Remarks	RAM can be accessed either directly (using common C commands like <code>X = Y;</code>) or indirectly. But indirect access using the FSR register is not allowed. Due to security reasons all instructions using FSR are removed during Upload. To avoid unintended behavior all constructions using addressing via FSR (either by the user or by the compiler) should be omitted. Instead of this IQRF OS provides complete support for indirect (byte oriented) addressing using extra system functions <code>readFromRAM</code> and <code>writeToRAM</code> . See example E06–RAM [10].
Side effects	–
See also	<code>writeToRAM</code>
Example1	<pre>IRP = 0; X = readFromRAM(bufferCOM + 10); IRP = 1; Y = readFromRAM(bufferRF + 10);</pre>
Example2	<pre>// Not correct: IRP = 0; // To access 0x190 IRP must be set to 1 X = readFromRAM(0x190) - 1; // 0x90 is addressed instead of expected 0x190</pre>
Example3	<pre>// Correct: IRP = 1; X = readFromRAM(0x90) - 1; // Perfect X = readFromRAM(0x190) - 1; // Also works thanks to the compiler feature</pre>
Example4	<pre>// Not allowed. The compiler uses FSR in such cases. for (i=0; i<5; i++) { A = bufferRF[i]; ... }</pre>
Example5	<pre>// Correct for (i=0; i<5; i++) { A = readFromRAM(bufferRF + i); ... }</pre>

writeToRAM

Function	Write one byte to specified location in RAM
Purpose	Indirect access to RAM registers
Syntax	<code>void writeToRAM(addr, value)</code>
Parameters	<ul style="list-style-type: none"> • <code>uns8 addr</code>: memory location address. Unlike <code>copyMemoryBlock</code>, "short" address within respective bank pair (up to 0xFF) should be used (e.g. 0xC5 for both 0xC5 and 0x1C5). But using long addresses (up to 0x1FF, e.g. 0x1C5) also works because the compiler trims the overlapping bit to correct 8b value. See Preconditions as well. • <code>uns8 value</code>: value to be written
Return value	–
Output values	–
Preconditions	<ul style="list-style-type: none"> • Before using this function, the IRP bank select bit must correspond to the location of the register (IRP=0 for RAM bank 0 and 1, IRP=1 for RAM bank 2 and 3). See the PIC datasheet [8] (Memory organization) and RAM map [2] for details. • Avoid writing to RAM areas dedicated to OS and to PIC special function registers otherwise OS can collapse. See RAM map [2]. • Some locations are restricted from writing. In doubt, refer to IQRF support by the manufacturer [6].
Remarks	RAM can be accessed either directly (using common C commands like <code>X = Y;</code>) or indirectly. But indirect access using the FSR register is not allowed. Due to security reasons all instructions using FSR are removed during Upload. To avoid unintended behavior all constructions using addressing via FSR (either by the user or by the compiler) should be omitted. Instead of this IQRF OS provides complete support for indirect (byte oriented) addressing using extra system functions <code>readFromRAM</code> and <code>writeToRAM</code> . See example E06–RAM [10]. Certain RAM locations are not accessible at all for security reasons.
Side effects	–
See also	<code>readFromRAM</code> , <code>copyMemoryBlock</code>
Example1	<pre>IRP = 0; writeToRAM(bufferCOM + 10, 2 * X); IRP = 1; writeToRAM(bufferRF + 10, 2 * X);</pre>
Example2	<pre>// Not correct: IRP = 0; // To access 0x190 IRP must be set to 1 writeToRAM(0x190, 201); // 0x90 is addressed instead of expected 0x190</pre>
Example3	<pre>// Correct: IRP = 1; writeToRAM(0x90, 201); // Perfect writeToRAM(0x190, 201); // Also works thanks to the compiler feature</pre>
Example4	<pre>// Not allowed. The compiler uses FSR in such cases. for (i=0; i<5; i++) bufferRF[i] = i;</pre>
Example5	<pre>// Correct for (i=0; i<5; i++) writeToRAM(bufferRF + i, i);</pre>

Buffers, data blocks

All functions for copying buffers (`copyBufferINFO2RF`, `copyBufferINFO2COM`, `copyBufferRF2COM`, `copyBufferRF2INFO`, `copyBufferCOM2RF`, `copyBufferCOM2INFO`) can use **offsets** `memoryOffsetFrom` and `memoryOffsetTo`. Offsets are applied when at least one of them is different from zero only. Then the following principle will take place: `memoryOffsetFrom` specifies relative offset in the From buffer and `memoryOffsetTo` specifies relative offset in the To buffer. It means that data is not read starting from `bufferXX[0]` but from `bufferXX[memoryOffsetFrom]` and is not stored starting from `bufferYY[0]` but from `bufferYY[memoryOffsetTo]`. Just the final part of the `bufferXX` is copied (from `memoryOffsetFrom` up to the end of the `bufferXX` or `bufferYY`, whichever is reached first).

If both `memoryOffsetFrom = 0` and `memoryOffsetTo = 0` complete buffers are copied. Offsets are default disabled (cleared after reset as well as after every buffer copy).

The `copyMemoryBlock` function can work with offsets as well but there is no reason to use them.

copyBufferINFO2COM

Function	Copy <code>bufferINFO</code> to <code>bufferCOM</code>
Purpose	Data transfer between buffers
Syntax	<code>void copyBufferINFO2COM()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
Remarks	<ul style="list-style-type: none"> • See example E06 - RAM [10]. • If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 35 B is copied.
Side effects	–
See also	<code>clearBufferINFO</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
Example1	<code>copyBufferINFO2COM();</code>
Example2	<pre>memoryOffsetFrom = 0; // bufferINFO to be copied memoryOffsetTo = 10; // to bufferCOM starting from bufferCOM[10]. copyBufferINFO2COM; // Just first 25 B is copied (until bufferCOM full).</pre>

copyBufferINFO2RF

Function	Copy <code>bufferINFO</code> to <code>bufferRF</code>
Purpose	Data transfer between buffers
Syntax	<code>void copyBufferINFO2RF()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
Remarks	<ul style="list-style-type: none"> • See example E06 - RAM [10]. • If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 35 B is copied.
Side effects	–
See also	<code>clearBufferINFO</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
Example	<code>copyBufferINFO2RF();</code>

copyBufferRF2COM

Function	Copy <code>bufferRF</code> to <code>bufferCOM</code>
Purpose	Data transfer between buffers
Syntax	<code>void copyBufferRF2COM()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
Remarks	<ul style="list-style-type: none"> • See example E06 - RAM [10]. • If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 35 B is copied.
Side effects	–
See also	<code>clearBufferINFO</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
Example	<code>copyBufferRF2COM();</code>

copyBufferRF2INFO

Function	Copy <code>bufferRF</code> to <code>bufferINFO</code>
Purpose	Data transfer between buffers
Syntax	<code>void copyBufferRF2INFO()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
Remarks	<ul style="list-style-type: none"> • See example E06 - RAM [10]. • If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 35 B is copied.
Side effects	–
See also	<code>clearBufferINFO</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2COM</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
Example	<code>copyBufferRF2INFO();</code>

copyBufferCOM2RF

Function	Copy <code>bufferCOM</code> to <code>bufferRF</code>
Purpose	Data transfer between buffers
Syntax	<code>void copyBufferCOM2RF()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
Remarks	<ul style="list-style-type: none"> • See example E06 - RAM [10]. • If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 35 B is copied.
Side effects	–
See also	<code>clearBufferINFO</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
Example	<code>copyBufferCOM2RF();</code>

copyBufferCOM2INFO

Function	Copy <code>bufferCOM</code> to <code>bufferINFO</code>
Purpose	Data transfer between buffers
Syntax	<code>void copyBufferCOM2INFO ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
Remarks	<ul style="list-style-type: none"> • See example E06 - RAM [10]. • If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 35 B is copied.
Side effects	–
See also	<code>clearBufferINFO</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyMemoryBlock</code>
Example	<code>copyBufferCOM2INFO () ;</code>

compareBufferINFO2RF

Function	Compare <code>bufferINFO</code> and <code>bufferRF</code> with respect to specified length
Purpose	Buffer comparison
Syntax	<code>bit compareBufferINFO2RF (length)</code>
Parameters	<code>uns8 length</code> : number of bytes to be compared (1 to 35)
Return value	<ul style="list-style-type: none"> • 1 – match • 0 – mismatch
Output values	–
Preconditions	–
Remarks	<ul style="list-style-type: none"> • See example E06 - RAM [10]. • If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 35 B is copied.
Side effects	–
See also	<code>clearBufferINFO</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2INFO</code> , <code>swapBufferINFO</code>
Example	<pre>if (!compareBufferINFO2RF(32)) // Compare 32 B then Error = 1; // Error if mismatch</pre>

copyMemoryBlock

Function	Copy specified RAM block to specified location
Purpose	Copy memory block within RAM
Syntax	<code>void copyMemoryBlock (from, to, length)</code>
Parameters	<ul style="list-style-type: none"> • <code>uns16 from</code>: starting address of the block to be copied • <code>uns16 to</code>: destination address • <code>uns8 length</code>: block length in bytes
Return value	–
Output values	–
Preconditions	<ul style="list-style-type: none"> • Copying is allowed also between different memory banks but both blocks are not allowed to cross bank boundaries (0x7F-0x80, 0xFF-0x100, 0x17F-0x180). • Unlike <code>writeToRAM</code>, "long" addresses (up to 0x1FF) are used. IRP is no object. • Avoid writing to RAM areas dedicated to OS and to PIC special function registers otherwise OS can collapse. See RAM map [2]. Some locations are restricted from writing due to security reasons..
Remarks	<ul style="list-style-type: none"> • <i>This function can access RAM within the whole memory area.</i> • It can work with offsets as well but there is no reason to use them. • See RAM map [2] and example E06 - RAM [10].
Side effects	–
See also	<code>writeToRAM</code>
Example	<pre>copyMemoryBlock(0x15D, 0x1DD, 4); // copy 4B block from 0x15D to 0x1DD copyMemoryBlock(bufferRF+10, bufferCOM+1, 8); // 8 bytes copied: // bufferCOM[1] = bufferRF[10] ... bufferCOM[8] = bufferRF[17]</pre>

swapBufferINFO

Function	Swap <code>bufferINFO</code> and <code>bufferAUX</code>
Purpose	Temporary <code>bufferINFO</code> saving
Syntax	<code>void swapBufferINFO ()</code>
Parameters	–
Return value	–
Output values	Content of <code>bufferINFO</code> and <code>bufferAUX</code> is swapped. See example E06 - RAM [10].
Preconditions	–
Remarks	32 B are swapped
Side effects	–
See also	<code>moduleInfo</code> , <code>appInfo</code>
Example	<pre>swapBufferInfo(); // Temporarily save bufferInfo to bufferAUX appInfo(); // Get user data from EEPROM ... swapBufferInfo(); // and restore previous data in bufferInfo</pre>

clearBufferINFO

Function	Clear <code>bufferINFO</code>
Purpose	<code>bufferINFO</code> clearing
Syntax	<code>void clearBufferINFO ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	Complete <code>bufferINFO</code> (35 B) is cleared (filled with zeros). See example E06 - RAM [10].
Side effects	–
See also	<code>copyBufferINFO2COM</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code> , <code>swapBufferINFO</code>
Example	<code>clearBufferINFO ();</code>

clearBufferRF

Function	Clear <code>bufferRF</code>
Purpose	<code>bufferRF</code> clearing
Syntax	<code>void clearBufferRF ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	Complete <code>bufferRF</code> (64 B) is cleared (filled with zeros). See example E06 - RAM [10].
Side effects	–
See also	<code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferINFO2RF</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
Example	<code>clearBufferRF ();</code>

moduleInfo

Function	Store Module data to <code>bufferINFO</code>								
Purpose	Get information about transceiver module and OS								
Syntax	<code>void moduleInfo ()</code>								
Parameters	-								
Return value	-								
Output values	<code>bufferINFO[0 to 7]</code>								
Preconditions	-								
Remarks	address in <code>bufferInfo</code>	7	6	5	4	3	2	1	0
	meaning	OS build		PIC type	OS version	Coordinator / Node	serial number		
	Module ID								
See IQRF OS User's guide [1] (Identification) for Module data format details.									
Side effects	-								
See also	<code>appInfo</code>								
Example	<pre> uns24 SN @ bufferInfo; uns8 OSv @ bufferInfo[4]; moduleInfo(); // Now SN == module serial number // and OSv == OS version </pre>								

appInfo

Function	Store Application data from EEPROM to <code>bufferINFO</code>								
Purpose	Get information about user application								
Syntax	<code>void appInfo ()</code>								
Parameters	-								
Return value	-								
Output values	<code>bufferINFO[0 to 31]</code>								
Preconditions	-								
Remarks	See IQRF OS User's guide [1] (Identification and Appendix, Memory maps).								
Side effects	-								
See also	<code>moduleInfo</code>								
Example1	<pre> appInfo(); // Copy Application data from EEPROM to bufferINFO copyBufferINFO2RF(); // and then to bufferRF </pre>								
Example2	<pre> #pragma packedCdataStrings 0 // Application data to EEPROM after compilation #pragma cdata[__EEAPPINFO] = "Application data, I'm user #01 " bufferINFO[0] = '2'; // Dynamic change of application data eeWriteData(__EEAPPINFO+29,1); // #01 changed to #02 appInfo(); // "Application data, I'm user #02 " is read </pre>								

SPI
enableSPI

Function	Activate SPI communication module and related pins
Purpose	Enable SPI communication
Syntax	<code>void enableSPI ()</code>
Parameters	–
Return value	–
Output values	SPI Status is switched to <i>SPI ready, communication mode</i> .
Preconditions	–
Remarks	<ul style="list-style-type: none"> • The PIC internal SPI hardware module and appropriate pins (C5 – C8) are configured and activated as SPI Slave. • See SPI Implementation in IQRF TR modules [5] and example E07-SPI [10].
Side effects	Related pins can not be used as general I/Os until SPI is disabled via <code>disableSPI</code> .
See also	<code>disableSPI</code> , <code>startSPI</code> , <code>stopSPI</code> , <code>getStatusSPI</code> , <code>restartSPI</code>
Example	See <code>getStatusSPI</code>

disableSPI

Function	Switch SPI HW module off and configure SPI pins as I/Os
Purpose	Disable SPI communication
Syntax	<code>void disableSPI ()</code>
Parameters	–
Return value	–
Output values	SPI Status is switched to <i>SPI not active</i> .
Preconditions	–
Remarks	The PIC internal SPI hardware module is disabled and related pins (C5 – C8) are reconfigured as general I/Os. See SPI Implementation in IQRF TR modules [5] and example E07-SPI [10].
Side effects	<ul style="list-style-type: none"> • The appropriate PIC pins are not restored to the state before <code>enableSPI</code> calling. • Current packet is lost by both sides if SPI communication is running on background at this moment.
See also	<code>enableSPI</code> , <code>startSPI</code> , <code>stopSPI</code> , <code>getStatusSPI</code> , <code>restartSPI</code>
Example	See <code>getStatusSPI</code>

startSPI

Function	Indicate ready to Master.
Purpose	<ul style="list-style-type: none"> Initiate SPI packet transmission from Slave (request to Master). Provide data from <code>bufferCOM</code> to Master according to Master's clock (on OS background). <code>startSPI(0)</code> indicates to Master that the Slave is ready to receive data (<code>bufferCOM</code> not full).
Syntax	<code>void startSPI(length)</code>
Parameters	<code>uns8 length</code> : number of bytes to be sent (0 to 41)
Return value	–
Output values	SPI Status is switched to: <ul style="list-style-type: none"> <i>SPI data ready</i> – after <code>startSPI(1 to 41)</code> <i>SPI ready, Communication mode</i> – after <code>startSPI(0)</code>.
Preconditions	SPI must be enabled by the <code>enableSPI</code> function before.
Remarks	<ul style="list-style-type: none"> SPI runs on OS background. <code>startSPI(0)</code> is also useful for recovering SPI from communication failures (e.g. the CRC mismatch). See SPI Implementation in IQRF TR modules [5] and example E07-SPI [10].
Side effects	–
See also	<code>enableSPI</code> , <code>disableSPI</code> , <code>stopSPI</code> , <code>getStatusSPI</code> , <code>restartSPI</code>
Example1	<pre> // Slave -> Master bufferCOM[0] = "I"; bufferCOM[1] = "Q"; enableSPI(); startSPI(2); // Request to Master is active on background from now ... // and the program just continues here </pre>
Example2	<code>startSPI(0);</code> // Reset SPI communication
Example3	See <code>getStatusSPI</code>

stopSPI

Function	Stop SPI communication
Purpose	Suspend SPI transmissions whenever it suits to Slave
Syntax	void stopSPI ()
Parameters	–
Return value	–
Output values	SPI Status is switched to <i>User stop</i> .
Preconditions	–
Remarks	<ul style="list-style-type: none"> • stopSPI is useful e.g. to avoid violation during preparation data to bufferCOM. • SPI transmission is stopped but SPI remains active (enabled). Communication can continue after next startSPI. • stopSPI is not needed after successful SPI reception to protect data received in bufferCOM. Data is protected by OS (and SPI status stays in mode 3F) until the slave allows further communication e.g. by the startSPI (0). • startSPI and stopSPI are not fully complementary. Receiving is allowed just after enableSPI without previous startSPI, startSPI is meaningful after previous startSPI not followed by stopSPI etc. • See SPI Implementation in the IQRF TR modules [5] and example E07-SPI [10].
Side effects	Current packet is lost by both sides if SPI communication is running on background at this moment.
See also	enableSPI, disableSPI, startSPI, getStatusSPI, restartSPI
Example	<pre> if (!getStatusSPI()) // If SPI is not in progress { stopSPI(); // Prohibit Master from transmitting bufferCOM[0] = ... // Prepare data to send bufferCOM[1] = ... startSPI(2); // Request to send } </pre>

restartSPI

Function	Indicate ready to continue SPI transfer to Master .
Purpose	Allow to continue SPI transmission (request to Master).
Syntax	void restartSPI ()
Parameters	–
Return value	–
Output values	
Preconditions	Intended after preceding stopSPI.
Remarks	SPI can continue from the state just before stopSPI.
Side effects	–
See also	startSPI, stopSPI
Example1	<pre> startSPI(16); // SPI started ... stopSPI(); // SPI stopped temporarily ... // to make some operations restartSPI(); // and allow to continue </pre>

getStatusSPI

Function	Update SPI flags and packet length and check whether SPI is busy
Purpose	Provide application program with information about current SPI status
Syntax	bit <code>getStatusSPI ()</code>
Parameters	–
Return value	<ul style="list-style-type: none"> • 1 – SPI busy • 0 – SPI not busy
Output values	<ul style="list-style-type: none"> • SPIpacketLength: received packet length • param2.3 (_SPIRX): 1 – Something received on SPI. • param2.4 (_SPICRCok): 1 – The last received SPI CRCM was O.K.
Preconditions	SPI must be enabled by <code>enableSPI</code>
Remarks	<ul style="list-style-type: none"> • Output values (param2) has different format than SPI status sent to the Master. • See SPI Implementation in IQRF TR modules [5] and example E07-SPI [10].
Side effects	–
See also	<code>enableSPI</code> , <code>disableSPI</code> , <code>startSPI</code> , <code>stopSPI</code> , <code>restartSPI</code>
Example1	<pre> // Master -> Slave enableSPI(); // Master is allowed to transmit from now Receive: clrwdt(); if (getStatusSPI()) // Wait until SPI is not busy goto Receive; if (_SPIRX) // Anything received? { // Yes: if (!_SPICRCok) // CRCM matched? { // No: startSPI(0); // Restart SPI goto Receive; // and try to receive again. } // Yes: // BufferCOM is automatically protected now // not to be overwritten by next SPI packet. // Thus, stopSPI is not necessary here. // Packet length is in SPIpacketLength. copyBufferCOM2INFO(); // Store received packet startSPI(0); // and then allow Master to transmit again. } else goto Receive; // Nothing received yet // ... Continue here after successful receiving waitMS(1); // Time for finishing startSPI(0) on background disableSPI(); // otherwise Master's CRCS check fails. // The delay depends on Master application. </pre>
Example2	<pre> enableSPI(); startSPI(2); // 2 B to send to master while (getStatusSPI()) // Wait until SPI is not busy waitMS(1); ... // Now the transfer is finished </pre>

RF
setTXpower

Function	Set RF output power
Purpose	Change RF range
Syntax	<code>void setTXpower (level)</code>
Parameters	uns8 level: 0 (min.) to 7 (max. – default) See datasheet of TR module, Table 2.
Return value	–
Output values	–
Preconditions	–
Remarks	–
Side effects	–
See also	RFTXpacket
Example	<code>setTXpower (7); // Max. RF output power</code>

setRFspeed

Function	Select RF bit rate
Purpose	Select RF bit rate
Syntax	<code>void setRFspeed (speed)</code>
Parameters	uns8 speed: <ul style="list-style-type: none"> • 1 1.2 kb/s (preliminary) • 2 19.2 kb/s (default) • 3 57.6 kb/s (preliminary) • 4 86.2 kb/s (preliminary)
Return value	–
Output values	–
Preconditions	–
Remarks	<ul style="list-style-type: none"> • Non-default bit rates are provisionally intended for experimental purposes only. • Routing is supported for 19.2 kb/s only
Side effects	RF channel must be specified after every bit rate change.
See also	setRFchannel
Example1	<code>setRFspeed (1); // 1.2 kb/s selected</code> <code>setRFchannel (...); // channel must be selected then</code>
Example2	<code>setRFspeed (2); // 19.2 kb/s selected</code> <code>setRFchannel (...); // channel must be selected then</code>

setRFband

Function	Select RF frequency band
Purpose	Select 868 MHz or 916 MHz band
Syntax	<code>void setRFband (band)</code>
Parameters	uns8 band: • 0 868 band MHz (default) • 1 916 band MHz
Return value	–
Output values	–
Preconditions	–
Remarks	Default channel is set (52 for 868 MHz band or 104 for 916 MHz band).
Side effects	RF channel must be specified after every band change.
See also	setRFchannel
Example1	<code>setRFband(1); // 916 MHz band selected</code>
Example2	<code>setRFband(0); // 868 MHz band selected</code>

setRFchannel

Function	Set RF channel
Purpose	Select free RF channel for not interfered communication
Syntax	<code>void setRFchannel (channel)</code>
Parameters	uns8 channel: see IQRF OS User's guide, Appendix 2, Channel map
Return value	–
Output values	–
Preconditions	–
Remarks	–
Side effects	RF channel must be specified after every bit rate or band change.
See also	setRFspeed
Example	<code>setRFband(0); // 868 MHz band selected</code> <code>setRFspeed(3); // 57.6 kb/s bit rate selected</code> <code>setRFchannel(25); // 868.15 MHz channel selected</code>

setRFmode

Function	Set RF mode
Purpose	Specify power management and signal filtering modes for RF transmission and receipt
Syntax	<code>void setRFmode (mode)</code>
Parameters	<p>uns8 mode: SWTTFRR in binary</p> <ul style="list-style-type: none"> • S: Stay in RX mode (for fast response for following <code>checkRF</code>, <code>RFRXpacket</code> or <code>RFTXpacket</code>) <ul style="list-style-type: none"> 1 RX chain stays enabled after <code>RFRXpacket</code> and <code>RFTXpacket</code> 0 RX chain is disabled after <code>RFRXpacket</code> and <code>RFTXpacket</code> • W: Wait packet end <ul style="list-style-type: none"> 1 Waits until receipt is finished if it is actually started even though <code>toutRF</code> timeout is over meanwhile. 0 <code>RFRXpacket</code> is unconditionally finished when <code>toutRF</code> timeout is over. • TT: TX mode <ul style="list-style-type: none"> 00 for STD RX mode (standard preamble ~3 ms) 01 for LP RX mode (prolonged preamble ~30 ms) 10 for XLP RX mode (prolonged preamble ~750 ms) 11 reserved • FF: Filter incoming signal in LP, XLP and RFIM RX modes (<code>RR</code> ≠ 0). Signal with lower level is ignored. Relative RF range is shortened due to this filtration. The level corresponds to the <code>checkRF(x)</code> parameter: <ul style="list-style-type: none"> 00 x = 5 01 x = 20 10 x = 35 11 x = 50 • RR: RX mode <ul style="list-style-type: none"> 00 STD RX mode (Standard, transmitting device should have <code>TT=00</code>) 01 LP RX mode (Low power, transmitting device should have <code>TT=01</code>) 10 XLP RX mode (Extra low power, transmitting device should have <code>TT=10</code>) 11 RFIM mode (<code>RFRXpacket</code> is terminated when signal strength falls below the <code>FF</code> level)
Return value	–
Output values	–
Preconditions	Non-STD RX modes are intended for bit rate 19.2 kb/s only.
Remarks	Default value is mode = 0. See example E10-RFMODE.
Side effects	RF circuitry and MCU is temporarily set to sleep during low power RX modes. Thus, all tasks running on OS background can be untimely canceled. To avoid this, use <code>setRFmode</code> after finishing background tasks. See Example 2.
See also	<code>checkRF</code>
Example1	<pre>setRFmode(0b00000000); // RX: STD, no filtering // TX: for STD RX (standard preambles) setRFmode(0b00010001); // RX: LP, lowest filtering (5) // TX: for LP RX (prolonged preambles ~3 ms) setRFmode(0b00101110); // RX: XLP, highest filtering (50) // TX: for XLP RX (prolonged preambles ~30 ms) setRFmode(0b00000101); // RX: LP, low filtering (20) // TX: for STD RX (standard preambles) setRFmode(0b10001011); // RX: RFIM, high filtering (35), stay in RX mode // TX: for STD RX (standard preambles) setRFmode(0b01000000); // RX: STD, no filtering, wait packet end // TX: for STD RX (standard preambles)</pre>
Example2	<pre>while (getStatusSPI()) // wait for finishing SPI on background clrwdt(); disableSPI(); SWDTEN = 0; // possibly disable watchdog for lower consumption setRFmode(0b00010001); // and go to LP mode then</pre>

checkRF

Function	Check incoming RF signal strength for specified level.
Purpose	Incoming RF signal detection to start RF receiving.
Syntax	<code>bit checkRF(level)</code>
Parameters	<p><code>uns8 level = DQI + RSSI_FILTER</code></p> <ul style="list-style-type: none"> • <code>DQI</code> (Data Quality Indicator, see the RF IC datasheet): <ul style="list-style-type: none"> • <code>0x80</code> DQI enabled • <code>0</code> DQI disabled • <code>RSSI_FILTER</code>: 0 to 64 <p>Higher level requires stronger signal. Relative RF range is shortened due to this filtration according the datasheet of the TR module, Table 3. RSSI offset is 32, e.g. level 16 means a signal with RSSI > 48.</p>
Return value	<ul style="list-style-type: none"> • 0: Signal with specified level or higher not detected RSSI < RSSI_FILTER, with respect to possible DQI • 1: Signal with specified level or higher detected RSSI > RSSI_FILTER, with respect to possible DQI
Output values	Signal strength is also available as a relative value in the <code>ADRESH</code> (one of PIC SFR registers). Higher value means stronger signal.
Preconditions	This function is intended for STD and RFIM receive modes but not for LP and XLP.
Remarks	<ul style="list-style-type: none"> • Higher level means lower sensitivity which requires stronger signal resulting in higher immunity against interferences but allows lower range – see TR datasheet, table Relative RF range vs. level. • Checking takes 1 ms or ~200 µs if RX chain is on (bit <code>S = 1</code> in <code>setRFmode</code>) • Checking consumes ~9.5 mA. • This function is intended for fast response and power consumption reduction in STD RX mode.
Side effects	–
See also	<code>setRFmode</code> , <code>RFRXpacket</code>
Example1	<pre> // Fast response receiving in STD mode if (checkRF(5)) // Detect signal with RSSI > 37 { if (RFRXpacket()) // Duration according to toutRF only if packet is sent. { // toutRF can be optimized for expected packet length. ... } } // Otherwise only ~1 ms is spent. ... time-critical section can be placed here </pre>
Example2	<pre> if (checkRF(10)) // Detect signal with RSSI > 42 ... </pre>
Example3	<pre> // RF signal strength analyzer SWDTEN = 0; // disable watchdog while (1) if (checkRF(3)) pulseLEDR(); // LED flash if signal level >= 3 detected </pre>

RFTXpacket

Function	Send RF packet of specified length from <code>bufferRF</code> .
Purpose	RF transmission
Syntax	<code>void RFTXpacket ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	<ul style="list-style-type: none"> • Peer-to-peer topology: <ul style="list-style-type: none"> • PIN = 0 (Peer-to-peer) • DLEN = packet length in bytes (0 to 64) • Prepare data to send in <code>bufferRF[0]</code> to <code>bufferRF[DLEN - 1]</code> (if <code>DLEN ≠ 0</code>) • Set RF output power via <code>setTXpower</code> • IQMESH: <ul style="list-style-type: none"> • PIN = 0x80 (IQMESH) • Other network related parameters should also be specified See IQRF OS User's guide [1] and IQMESH specification [4].
Remarks	<ul style="list-style-type: none"> • Unlike SPI, RF communication does not run on OS background. This function is active on foreground until the packet is sent. • Duration depends on TR type, routing algorithm, packet length and timeslot. • See examples E01–TX, E03–TR, E09–LINK [10] and E11–IQMESH-C [10]. • Only one stack level is available to call this function in subroutine.
Side effects	<ul style="list-style-type: none"> • <code>bufferRF[DLEN]</code> and <code>bufferRF[DLEN+1]</code> are destroyed • The RF circuitry wakes up (in case of sleeping).
See also	<code>RFRXpacket</code> , <code>setTXpower</code> , <code>setRFmode</code> and (in case of IQMESH) also other RF functions

Example1	<pre> // Peer-to-peer topology PIN=0; // Peer-to-peer (update also after every RFRXpacket // before every RFTXpacket) setNonetMode(); bufferRF[0] = "I"; // Data to send bufferRF[1] = "Q"; DLEN = 2; // 2 B packet RFTXpacket(); // Send the packet to all Peer-to-peer Nodes in range // and to all IQMESH Nodes having set filtering off // Program stays here until the packet is sent ... // and then continues </pre>
Example2	<pre> // IQMESH without routing, packet from Coordinator to Node #10 PIN = 0; // PIN preclearing (update also after every RFRXpacket // before every RFTXpacket) setCoordinatorMode(); // The NTWF flag (PIN.7) is set here. bufferRF[0] = "I"; // Data to send bufferRF[1] = "Q"; DLEN = 2; // 2 B packet RX = 10; // Packet for Node #10 // _ROUTEF = 0; // Routing disabled - not necessary (default by OS) RFTXpacket(); // Send the packet to IQMESH Node #10 in this network // Reception depends on the Node (its current network // or filtering) </pre>
Example3	<pre> // IQMESH with routing // Packet from Coordinator to Node #10 PIN = 0; // PIN preclearing (update also after every RFRXpacket // before every RFTXpacket) setCoordinatorMode(); // The NTWF flag (PIN.7) is set here. bufferRF[0] = "I"; // Data to send bufferRF[1] = "Q"; DLEN = 5; // 5 B packet RX = 10; // Packet for Node #10 _ROUTEF = 1; // Routing enabled for outgoing packets RTDEF = 1; // SFM (Static Full MESH) // RTDEF = 2; // DFM (Discovered Full MESH) RTDT0 = 10; // 10 hops // RTDT0 = eeReadByte[0]; // # hops = # bonded nodes RTDT1 = 2; // Time slot = 2 ticks (20 ms is enough for DLEN=5) RFTXpacket(); // Send the packet to IQMESH Node #10 in this network // Reception depends on the Node (its current network // or filtering) </pre>

RFRXpacket

Function	Receive RF packet to <code>bufferRF</code> and provide related information
Purpose	RF receiving
Syntax	bit <code>RFRXpacket ()</code>
Parameters	–
Return value	<ul style="list-style-type: none"> • 1 – packet received • 0 – packet not received
Output values	<ul style="list-style-type: none"> • <code>lastRSSI</code> – the RSSI value after successful receipt (single sample). Quiet level (noise) is 25 - 28. • <code>DLEN</code> = packet length. This variable is destroyed if the receipt is not successful. • <code>PIN</code> is updated according to packet received. This variable is destroyed if the receipt is not successful. • <code>_NTWPACKET</code>: valid if <code>RFRXpacket</code> return value == 1 only: <ul style="list-style-type: none"> • 1 – networking packet received • 0 – non-networking packet received • Other related networking information in case of IQMESH.
Preconditions	<ul style="list-style-type: none"> • Timeout should be specified in <code>toutRF</code> (1 to 255) in number of 10 ms ticks or for LP and XLP modes in cycles (RFIC On-Off period – see IQRF OS User's guide, RF modes). • Peer-to-peer topology: nothing else • IQMESH: network related parameters (filtering, ...) should be predefined See IQRF OS User's guide [1] and IQMESH specification [4].
Remarks	<ul style="list-style-type: none"> • Unlike SPI, RF communication does not run on OS background. This function is active on foreground until the packet is received or timeout expired. Timeout during packet receiving terminates the reception except of the Wait packet end mode – see <code>setRFmode</code>. • If the packet is sent when the address (or a routing device) is not executing this function the packet is lost. • Peer-to-peer topology: All packets in range are received. • IQMESH: Device receives only packets intended for it and non-networking packets depending on filtering mode – see <code>setNetworkFilteringOn</code> and <code>setNetworkFilteringOff</code>. • See examples E02–RX, E03–TR, E09–LINK and E11-IQMESH-N [10]. • Only one stack level is available to call this function in subroutine.
Side effects	<ul style="list-style-type: none"> • Update <code>PIN</code> before every <code>RFTXpacket</code> folowed after <code>RFRXpacket</code>. • Result of <code>captureTicks</code> is destroyed if <code>startCapture</code> is active on background at the same time. • <code>bufferRF[DLEN]</code> and <code>bufferRF[DLEN+1]</code> is destroyed. • The RF circuitry wakes up (in case of sleeping). • In LP and XLP modes the Watchdog is disabled during this operation and enabled after finishing.
See also	<code>RFTXpacket</code> , <code>setRFmode</code> , <code>checkRF</code> and (in case of IQMESH) also other RF functions
Example1	<pre> // Peer-to-peer topology toutRF = 10; // RF timeout 100 ms if RFRXpacket (); // Try to receive RF packet. // Program stays here until the packet is received // or the timeout is expired. Packet received? { // Yes: copyBufferRF2INFO (); // Store received data PacketLength = DLEN; // and possibly other info (packet length, ...) } else { // No: ... // Timeout expired. Arrange respective operations. } </pre>
Example2	IQMESH: See <code>setNodeMode</code> and <code>setNetworkFilteringOn</code> .

Networking

setCoordinatorMode

Function	Set Coordinator mode
Purpose	Assign the TR module as a network Coordinator
Syntax	<code>void setCoordinatorMode ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH only
Remarks	Every TR module can work as a Coordinator or a Node. Just one Coordinator in single network is allowed. Avoid dynamic switching the Coordinator from device to device in a network. This settings affects both <code>RFRXpacket</code> and <code>RFTXpacket</code> .
Side effects	–
See also	<code>setNodeMode</code> , <code>setNonetMode</code>
Example	

setNodeMode

Function	Set Node mode
Purpose	Assign the TR module as a network Node
Syntax	<code>void setNodeMode ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH only
Remarks	Every TR module can work as a Coordinator or a Node. This settings affects both <code>RFRXpacket</code> and <code>RFTXpacket</code> .
Side effects	–
See also	<code>setCoordinatorMode</code> , <code>setNonetMode</code>
Example	

setNonetMode

Function	Select Peer-to-peer mode
Purpose	Switch from IQMESH to Peer-to-peer
Syntax	void setNonetMode ()
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	<ul style="list-style-type: none">• Default OS mode is Peer-to-peer.• This settings affects <code>RFRXpacket</code> and <code>RFTXpacket</code> features.• PIN is not affected immediately but it is cleared after subsequent <code>RFRXpacket</code> or <code>RFTXpacket</code>.
Side effects	–
See also	<code>setCoordinatorMode</code> , <code>setNodeMode</code>
Example	<pre>setNetworkOne(); // TR communicates in IQMESH networking mode here ... // setNonetMode(); // Switch to Peer-to-peer mode ... // Now TR communicates without networking support</pre>

setNetworkFilteringOn

Function	Start filtering incoming non-networking packets and packets coming from non-current network.
Purpose	To receive packets from current network only.
Syntax	<code>void setNetworkFilteringOn ()</code>
Parameters	–
Return value	–
Output values	This affects the RFRXpacket return value.
Preconditions	For IQMESH only. Default OS condition is Filtering Off.
Remarks	–
Side effects	–
See also	setNetworkFilteringOff, RFRXpacket
Example	<pre> setNetworkFilteringOn(); // Start filtering incoming packets RFRXpacket(); // Return value == 1 if the packet came // from current network only. // Return value == 0 if // the packet came from non-current network(s) // or it is a non-networking packet // or no packet came in time at all. </pre>

setNetworkFilteringOff

Function	Stop filtering incoming packets from the point of view the packet is coming from.
Purpose	To receive all packets (non-networking packets as well as packets from all network).
Syntax	<code>void setNetworkFilteringOff ()</code>
Parameters	–
Return value	–
Output values	This affects the RFRXpacket return value.
Preconditions	For IQMESH only. Default OS condition is Filtering Off.
Remarks	Network 1 or 2 is automatically selected according to last received packet in this mode (except of non-networking packets).
Side effects	–
See also	setNetworkFilteringOn, RFRXpacket
Example	<pre> setNetworkFilteringOff(); // Stop filtering incoming packets RFRXpacket(); // Return value == 1 if // the packet came from current network // or from non-current network(s) // or it is a non-networking packet // Return value == 0 if // no packet came in time at all </pre>

setUserAddress

Function	Assign a user address to a Node
Purpose	User addressing of Nodes
Syntax	<code>void setUserAddress (address)</code>
Parameters	uns16 address: user address 1 to 65 000
Return value	–
Output values	–
Preconditions	For IQMESH Node and DFM2B only.
Remarks	<ul style="list-style-type: none"> • 0xFFFF is intended for broadcast. • Groups can be created by assigning the same address to more Nodes. • See Routing algorithms in the IQRF OS user's guide for details. • It is often convenient to set this as a part of bonding procedure by the user (to keep user program the same for all Nodes etc.). • The Node has User address stored in RAM. Thus, this is lost after TR module reset.
Side effects	–
See also	bondNewNode
Example 1	<code>setUserAddress(2000); // The Node has got user address 2000</code>
Example 2	<pre>setUserAddress(2000); ... reset(); // User address is lost after reset setUserAddress(2000); // User address restored</pre>
Example 3	<pre>setUserAddress(UA); eeWriteByte(E EUA, UA) // User address stored to EEPROM ... reset(); // User address lost after reset setUserAddress(eeReadByte(E EUA)); // User address restored from EEPROM</pre>

getNetworkParams

Function	Get network parameters																
Purpose	Get information about the network																
Syntax	uns8 getNetworkParams ()																
Parameters	–																
Return value	<table border="1" style="margin-left: 20px;"> <thead> <tr> <th>7</th> <th>6</th> <th>5</th> <th>4</th> <th>3</th> <th>2</th> <th>1</th> <th>0</th> </tr> </thead> <tbody> <tr> <td>networkTwo</td> <td>networkingMode</td> <td>routingOn</td> <td>CoordinatorMode</td> <td>AUXB</td> <td>–</td> <td>–</td> <td>–</td> </tr> </tbody> </table> <p> networkTwo = 1 Parameters for Network 2 are used networkingMode = 1 Module works in networking topology routingOn = 1 Module routes in RFRXpacket CoordinatorMode = 1 Module works as a Coordinator AUXB Auxiliary, not intended for the user </p>	7	6	5	4	3	2	1	0	networkTwo	networkingMode	routingOn	CoordinatorMode	AUXB	–	–	–
7	6	5	4	3	2	1	0										
networkTwo	networkingMode	routingOn	CoordinatorMode	AUXB	–	–	–										
Output values	<ul style="list-style-type: none"> • param2: Address of current device in network (0 - 239). For unbonded device 0 is returned. • bit _NTWPACKET: 1 – IQMESH packet 0 – Peer-to-peer packet • param3: Network identification (param3.high=CLID1, param3.low=CLID0). If the device is bonded CLID0/1 refers to Coordinator otherwise to the device itself. These features are not guaranteed for future OS versions. 																
Preconditions	For IQMESH only.																
Remarks	See example E11 - IQMESH-N [10].																
Side effects	–																
See also	amIBonded																
Example	<pre> if (amIBonded()) // Is the Node bonded? { // Yes: getNetworkParams(); // Get Node number myAddr = param2; } </pre>																

Routing

setRoutingOn

Function	Routing enabled
Purpose	Allow the Node to route packets on background.
Syntax	<code>void setRoutingOn ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH Nodes only. Default OS condition is Routing on.
Remarks	<ul style="list-style-type: none"> • Routing must be enabled for a Node to be assigned to the routing backbone during Discovery. • Routing can be enabled in all receive modes (STD, LP, XLP and RFIM).
Side effects	–
See also	<code>setRoutingOff</code> , <code>discovery</code> , <code>isDiscoveredNode</code> , <code>wasRouted</code>
Example	–

setRoutingOff

Function	Routing disabled
Purpose	Forbid the Node to route packets on background.
Syntax	<code>void setRoutingOff ()</code>
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH Nodes only. Default OS condition is Routing on.
Remarks	If routing is disabled the Node will not be assigned to the routing backbone during Discovery.
Side effects	–
See also	<code>setRoutingOn</code> , <code>discovery</code> , <code>isDiscoveredNode</code> , <code>wasRouted</code>
Example	–

discovery

Function	Discover Nodes for routing and assign VRN (Virtual Routing Number) to individual Nodes
Purpose	Routing backbone creation (for routing transparent from the user's point of view)
Syntax	<code>uns8 discovery (zones)</code>
Parameters	<code>uns8: zones:</code> max. number of zones to be established (= max. number of hops allowed per packet)
Return value	Number of discovered Nodes (\leq number of bonded Nodes)
Output values	<ul style="list-style-type: none"> Routing backbone is stored in EEPROM
Preconditions	<ul style="list-style-type: none"> For IQMESH Coordinator only. Nodes must be in the <code>answerSystemPacket</code> loop routine during Discovery.
Remarks	<ul style="list-style-type: none"> Nodes in current network only are discovered. Discovery should be invoked after every change in network topology. Nodes use the TX output power currently set in Coordinator for discovery. (It is recommended to run <code>discovery</code> with lower RF output power than in normal communication.) See IQRF OS User's guide, routing algorithms. See example E11 - IQMESH-C [10].
Side effects	<ul style="list-style-type: none"> Watchdog is disabled during this operation and enabled after finishing. All OS buffers (<code>bufferINFO</code>, <code>bufferCOM</code>, <code>bufferRF</code> and <code>bufferAUX</code>) are destroyed
See also	<code>setRoutingOn</code> , <code>setRoutingOff</code> , <code>isDiscoveredNode</code> , <code>bondNewNode</code> , <code>answerSystemPacket</code>
Example1	<pre>setTXpower(DISCOVERY_POWER); // Set RF power for discovery nodes = discovery(10); // Limit to max. 10 hops SWDTEN = 0; // Possibly restore WDT</pre>
Example2	<pre>nodes = discovery(eeReadByte(0x00)); // Limit to number of bonded Nodes // e.g. for Chain MESH</pre>

answerSystemPacket

Function	Enable response to Coordinator for Discovery
Purpose	Discovery support from the Node's side
Syntax	<code>void answerSystemPacket ()</code>
Parameters	–
Return value	–
Output values	Routing information exchanged between Coordinator and the Node via system packets.
Preconditions	<ul style="list-style-type: none"> • For IQMESH Node only. • Nodes must be in the <code>answerSystemPacket</code> loop routine when Discovery is running. • WDT should be disabled before <code>answerSystemPacket</code>
Remarks	<ul style="list-style-type: none"> • Nodes use the TX output power currently set in Coordinator for discovery. It is recommended to run <code>discovery</code> with lower RF output power than in normal communication. See E11-IQMESH-N [10]. • No stack level is available to call this function in subroutine.
Side effects	<ul style="list-style-type: none"> • <code>toutRF</code> is changed after Discovery • RF output power is inherited from the Coordinator and not restored
See also	<code>setRoutingOn</code> , <code>setRoutingOff</code> , <code>isDiscoveredNode</code> , <code>discovery</code>
Example	<pre> toutRF= MY_TOUT_RF; if (RFRXpacket ()) { ... } else { if (lastRSSI > discovery_threshold) { SWDTEN = 0; answerSystemPacket (); // to be discovered SWDTEN = 1; setTXpower (MY_POWER); } } </pre>

isDiscoveredNode

Function	Check for being discovered
Purpose	Ask whether the Node has been discovered
Syntax	bit <code>isDiscoveredNode (address)</code>
Parameters	uns8: address: Node address
Return value	<ul style="list-style-type: none"> • true: Specified Node has been discovered • false: Specified Node has not been discovered
Output values	–
Preconditions	For IQMESH Coordinator only.
Remarks	See E11-IQMESH-C [10].
Side effects	–
See also	discovery, answerSystemPacket, optimizeHops
Example	<pre> DiscoveredNodes = discovery(3); // Discovery (up to 3 zones) if (DiscoveredNodes < BondedNodes) // (BondedNodes and DiscoveredNodes // are user variables) { // There are some bonded but not discovered Nodes if !isDiscoveredNode(1) // Is the Node 1 discovered? ... } else { // All bonded Nodes discovered ... } </pre>

wasRouted

Function	Indicate incoming packet routing
Purpose	To distinguish whether incoming packet has been routed for other recipient(s).
Syntax	bit <code>wasRouted ()</code>
Parameters	–
Return value	<ul style="list-style-type: none"> • true packet has been routed • false packet has not been routed
Output values	–
Preconditions	For IQMESH Nodes only.
Remarks	Addressees route broadcast packets only. See E11-IQMESH-N [10].
Side effects	–
See also	setRoutingOn, setRoutingOff, discovery, isDiscoveredNode
Example	<pre> if (RFRXpacket ()) { if (wasRouted()) pulseLEDG(); // indicate routing received packet for broadcast ... } else { if (wasRouted()) pulseLEDG(); // indicate routing incoming packet for another addressee } </pre>

optimizeHops

Function	Optimize number of hops for given Node
Purpose	Set optimized number of hops according to a topology, without flooding
Syntax	<code>void optimizeHops (x)</code>
Parameters	uns8 x: optimizing method <ul style="list-style-type: none"> • 0xFF DOM – Discovered optimized MESH: sets RTDT0 to VRN of addressed Node • 0x00 DRM – Discovered reduced MESH: sets RTDT0 to VRN of the first Node in the zone of the addressed Node. <i>Not implemented yet.</i>
Return value	–
Output values	RTDT0 (number of hops) is set
Preconditions	<ul style="list-style-type: none"> • For IQMESH Coordinator and DFM routing algorithm only. • Intended to be called before sending a packet from Coordinator. • Node address must be set before (RX = ...). • The Node must be discovered.
Remarks	See E11-IQMESH-C [10] and IQRF OS User's guide.
Side effects	–
See also	discovery, isDiscoveredNode
Example	<pre> setCoordinatorMode (); RX = MY_NODE; RTDT0 = eeReadByte(0x00); // Hops according to a number of bonded Nodes if (isDiscoveredNode(RX)) // For routing using Discovery only optimizeHops(0xFF); // Modifies RTDT0 (number of hops) : : RFTXpacket (); </pre>

Bonding – Node only

bondRequest

Function	Ask Coordinator via RF for bonding to its network. Bond the Node in cooperation with Coordinator and record it to EEPROM.
Purpose	Request by the Node to be included to the network on both Coordinator's and Node's sides.
Syntax	bit <code>bondRequest()</code>
Parameters	–
Return value	<ul style="list-style-type: none"> • 1 – Node has been bonded • 0 – Node has not been bonded
Output values	<ul style="list-style-type: none"> • The <code>amIBonded</code> function starts to return value == 1 whenever is called while the Node is bonded by <code>bondRequest</code> not being unbonded by <code>removeBond</code> or <code>wipeBondNR</code>. • Coordinator is not affected at all. • <code>param2</code>: Node address (if successfully bonded only). Not guaranteed for future OS versions.
Preconditions	For IQMESH only.
Remarks	<ul style="list-style-type: none"> • Bonding is a mutual relationship between Coordinator and Node. Coordinator assigns a Node number (1 to 239 or 0xEF) to the Node which serves as Node address within the network. (Coordinator itself has the address 0.) Bonding accomplishes via exchanging system RF packets and results are stored in system part of internal EEPROMs. The user can access results and change them via other functions related to bonding. See example E11 - IQMESH-N, E11 - IQMESH-C [10]. This function is active until successfully finished or fixed 10 s timeout expired. RF power is not affected. • No stack level is available to call this function in subroutine.
Side effects	<p>The following values are modified and not restored:</p> <ul style="list-style-type: none"> • DLEN • PIN • toutRF • <code>bufferRF[0 to 63]</code> destroyed • <code>bufferINFO[0 to 34]</code> destroyed • Result of <code>captureTicks</code> is destroyed if <code>startCapture</code> is active on background at the same time. • Watchdog is disabled during this operation and enabled after finishing
See also	<code>bondNewNode</code> , <code>amIBonded</code> , <code>removeBond</code> , <code>rebondNode</code> , <code>getNetworkParams</code>
Example1	<pre> pulsingLED(); // LED blinking indicates attempt to bond (max. 10 s) if (bondRequest()) { // if successfully bonded stopLED(); _RLED=1; // LED On waitDelay(100); // for 1 s } stopLED(); </pre>
Example2	See <code>amIBonded</code>

amIBonded

Function	Is the Node bonded?
Purpose	Test whether the Node is bonded on Node's side
Syntax	bit <code>amIBonded()</code>
Parameters	–
Return value	<ul style="list-style-type: none"> • 1 – Node is bonded (after <code>bondRequest</code> not being unbonded by <code>removeBond</code>) • 0 – Node is not bonded: <ul style="list-style-type: none"> • no <code>bondRequest</code> has ever been successfully executed • after <code>removeBond</code>
Output values	–
Preconditions	For IQMESH only. Result is not depended on the Coordinator at all.
Remarks	See example E11 - IQMESH-N [10].
Side effects	–
See also	<code>bondRequest</code> , <code>removeBond</code>
Example	<pre>while (!amIBonded()) // Request for being bonded (if not bonded yet) { bondRequest(); // Repeatedly try to bond clrwdt(); // until successful }</pre>

removeBond

Function	Remove the Node from the network and record it to EEPROM.
Purpose	Exclude the Node from the network on Node's side and keep its Node number reserved for possible future rebonding.
Syntax	void <code>removeBond()</code>
Parameters	–
Return value	–
Output values	<ul style="list-style-type: none"> • The <code>amIBonded</code> function starts to return value == 0 whenever is called until the Node is bonded again via <code>bondRequest</code>. • Just this value is affected but the Node keeps the Node number still stored (for possible future rebonding with the same Node number). • Coordinator is not affected at all.
Preconditions	For IQMESH only.
Remarks	<ul style="list-style-type: none"> • See example E11 - IQMESH-N [10]. • For rebonding use <code>bondRequest</code> again. • <code>removeBond</code> relates to Node only and <code>removeBondedNode</code> and <code>rebondNode</code> relate to Coordinator only. The other side is not informed by OS about changes made by these functions. If synchronization is needed it should be done by the application.
Side effects	<ul style="list-style-type: none"> • The effect of <code>removeBond</code> will not surface in Coordinator until <code>bondRequest/bondNewNode</code> procedure is invoked afterward. That is why Coordinator can use the number reserved for this Node for another Node meanwhile (after possible <code>removeBondedNode</code>). In this case the Coordinator will assign a different number.
See also	<code>bondRequest</code> , <code>bondNewNode</code> , <code>amIBonded</code> , <code>rebondNode</code>
Example	<code>removeBond(); // Remove the bond.</code>

Bonding – Coordinator only
bondNewNode

Function	Look for bond requesting devices and bond a new Node by Coordinator on a Node's request via RF. Allocate the Node number and assign the Network number and send both to Node via RF. If successful, the Node is bonded to the network on both Coordinator's and Node's sides and is included to the list of bonded Nodes provided by Coordinator in EEPROM.
Purpose	Include a new Node to the network
Syntax	bit bondNewNode (address)
Parameters	<p>uns8: address</p> <ul style="list-style-type: none"> • 1 to 239 Assign requested address to the Node. This must be unique in the whole network. If an existing number is used the Node is not bonded and the function immediately returns 0. Only these Nodes can be a part of routing backbone. • 0 The first free address is assigned (like the only way in IQRF OS v2.xx). It equals to a number of bonded nodes + 1. It assumes a continuous block of addresses and possible vacations are ignored. Thus, this way is suitable for the initial bonding without discontinuities. • 0xFE The universal address. Nodes with this address are included in the network but outside the routing backbone (not being discovered). Particular address can be assigned by <code>setUserAddress</code>. It is intended especially for networks with more than 239 Nodes.
Return value	<ul style="list-style-type: none"> • 1 – bonding successful, Node included to the list of bonded Nodes • 0 – bonding unsuccessful, Node not included to the list of bonded Nodes
Output values	<ul style="list-style-type: none"> • param2: Node number • bufferRF[0 to 1]: two lower ID bytes of the Node (is successfully bonded), LSB in bufferRF[0]. • The <code>isBondedNode</code> function starts to return value == 1 whenever is called while the Node is in the list of bonded Nodes.
Preconditions	<ul style="list-style-type: none"> • For IQMESH Coordinator only. • Coordinator accomplishes bonding on request from Node via RF. When this function is executing the <code>bondRequest</code> function must just be active in the Node.
Remarks	<ul style="list-style-type: none"> • See example E11 - IQMESH-C [10] and IQRF OS User's guide – routing algorithms. • If no requesting Node is detected during 10 s period this function terminates. • Network number is derived from Coordinator ID which ensures unique identification of various networks. • RF power is not affected. • An occupied address can be unblocked by <code>removeBondedNode (address)</code>. • No stack level is available to call this function in subroutine.
Side effects	<p>The following values are modified and not restored:</p> <ul style="list-style-type: none"> • toutRF modified • bufferRF[0 to 63] = destroyed • bufferINFO[0 to 34] = destroyed • Result of <code>captureTicks</code> is destroyed if <code>startCapture</code> is active on background at the same time. • Watchdog is disabled during this operation and enabled after finishing
See also	<code>bondRequest</code> , <code>removeBondedNode</code> , <code>rebondNode</code> , <code>isBondedNode</code> , <code>setUserAddress</code>
Example	<pre> if (bondNewNode ()) // Bonding successful ? { // Yes: NodeNumber = param2; ... } else { // No: ... // Arrange necessary steps } </pre>

isBondedNode

Function	Is specified Node in the list of bonded Nodes?
Purpose	Test whether the Node is bonded on Coordinator's side
Syntax	bit isBondedNode (n)
Parameters	uns8 n: Node number
Return value	<ul style="list-style-type: none"> • 1 – Node is in the list of bonded Nodes • 0 – Node is not in the list of bonded Nodes
Output values	–
Preconditions	For IQMESH only. The result is not affected by the Node at all.
Remarks	–
Side effects	–
See also	bondNewNode, removeBondedNode, rebondNode, clearAllBonds
Example	<pre> if isBondedNode(28) // Is Node #28 bonded ? { // Yes: ... // Coordinator assumes Node #28 to be bonded } else { // No: ... // Coordinator assumes Node #28 not to be bonded } </pre>

removeBondedNode

Function	Remove a Node from the list of bonded Nodes by Coordinator in EEPROM
Purpose	Exclude the Node from the network on Coordinator's side
Syntax	void removeBondedNode (n)
Parameters	uns8 n: Node number
Return value	–
Output values	The <code>isBondedNode</code> function starts to return value == 0 whenever is called while the Node is not in the list of bonded Nodes. The Node is not affected at all.
Preconditions	For IQMESH only
Remarks	<code>removeBondedNode</code> and <code>rebondNode</code> relate to Coordinator only and <code>removeBond</code> relates to Node only. The other side is not informed by OS about changes made by these functions. If synchronization is needed it should be done by the application.
Side effects	–
See also	bondNewNode, isBondedNode, clearAllBonds, removeBond
Example	<pre> removeBondedNode(28); // Coordinator assumes Node #28 to be // out of the network from now on </pre>

rebondNode

Function	Put a Node back to the list of bonded Nodes by Coordinator in EEPROM
Purpose	Include the Node to the network again on Coordinator's side
Syntax	<code>bit rebondNode (n)</code>
Parameters	<code>uns8 n</code> : Node number
Return value	reserved for future OS versions
Output values	The <code>isBondedNode</code> function starts to return value == 1 whenever is called while the Node is in the list of bonded Nodes. The Node is not affected at all.
Preconditions	For IQMESH only. Avoid rebonding a Node not being bonded ever before.
Remarks	<code>removeBondedNode</code> and <code>rebondNode</code> relate to Coordinator only and <code>removeBond</code> relates to Node only. The other side is not informed by OS about changes made by these functions. If synchronization is needed it should be done by the application.
Side effects	–
See also	<code>bondNewNode</code> , <code>removeBondedNode</code> , <code>isBondedNode</code>
Example	<pre>rebondNode(28); // Coordinator assumes Node #28 to be // back in the network from now on</pre>

clearAllBonds

Function	Remove all Nodes from the list of bonded Nodes by Coordinator in EEPROM
Purpose	Excluding all Nodes from the network on Coordinator's side
Syntax	<code>void clearAllBonds ()</code>
Parameters	–
Return value	–
Output values	The <code>isBondedNode</code> function starts to return value == 0 whenever is called while the Node is not in the list of bonded Nodes. Nodes are not affected at all.
Preconditions	For IQMESH only
Remarks	See example E11 - IQMESH-C [10]. Coordinator will start to assign Node numbers from 0.
Side effects	<code>bufferINFO[0 to 34]</code> destroyed
See also	<code>removeBondedNode</code>
Example	<pre>clearAllBonds(); // Exclude all currently bonded nodes from the network</pre>

Documentation and Information

- 1 IQRF OS User's guide www.iqrf.org/weben/downloads.php?id=155
- 2 **RAM map and EEPROM map**, IQRF OS User's guide, Appendix 1 [1]
- 3 **IQRF website** www.iqrf.org
- 4 **IQMESH specification** www.iqmesh.org/iqmesh
- 5 **SPI specification** www.iqrf.org/weben/downloads.php?id=85
- 6 **IQRF support site** www.iq-esupport.com
- 7 **TR-52B datasheet**: www.iqrf.org/weben/downloads.php?id=91
- 8 **PIC16F886 datasheet**: www.iqrf.org/weben/downloads.php?id=126
- 9 **IQRF IDE**: www.iqrf.org/weben/downloads.php?id=86
- 10 **Basic examples** (included in the StartUp Package): www.iqrf.org/weben/downloads.php?id=112

If you need a help or more information please visit IQRF support pages [6] and Submit a Ticket with your request. A lot of information is also available in the IQRF OS User's guide [1] and on the IQRF home page [3].

Document revision

- 110112 Information added and precised
- 101223 OS v3.0, first release for TR-52B, TR-53B and compatibles

Index

amIBonded.....	48	pulsingLEDR.....	13
answerSystemPacket.....	44	readFromRAM.....	18
applInfo.....	25	rebondNode.....	51
bondNewNode.....	49	removeBond.....	48
bondRequest.....	47	removeBondedNode.....	50
calibrateTimer.....	4	reset.....	4
captureTicks.....	10	restartSPI.....	28
checkRF.....	33	RFRXpacket.....	36
clearAllBonds.....	51	RFTXpacket.....	34
clearBufferINFO.....	24	setCoordinatorMode.....	37
clearBufferRF.....	24	setNetworkFilteringOff.....	39
compareBufferINFO2RF.....	22	setNetworkFilteringOn.....	39
copyBufferCOM2INFO.....	22	setNodeMode.....	37
copyBufferCOM2RF.....	21	setNonetMode.....	38
copyBufferINFO2COM.....	20	setOffPulsingLED.....	12
copyBufferINFO2RF.....	20	setOnPulsingLED.....	12
copyBufferRF2COM.....	21	setRFband.....	31
copyBufferRF2INFO.....	21	setRFchannel.....	31
copyMemoryBlock.....	23	setRFmode.....	32
debug.....	6	setRFready.....	6
disableSPI.....	26	setRFsleep.....	5
discovery.....	43	setRFspeed.....	30
eeReadByte.....	16	setRoutingOff.....	42
eeReadData.....	16	setRoutingOn.....	42
eeWriteByte.....	17	setTXpower.....	30
eeWriteData.....	17	setUserAddress.....	40
enableSPI.....	26	startCapture.....	9
getNetworkParams.....	41	startDelay.....	10
getStatusSPI.....	29	startLongDelay.....	11
getSupplyVoltage.....	7	startSPI.....	27
getTemperature.....	7	stopLEDG.....	15
iqrfSleep.....	5	stopLEDR.....	14
isBondedNode.....	50	stopSPI.....	28
isDelay.....	11	swapBufferINFO.....	23
isDiscoveredNode.....	45	waitDelay.....	8
moduleInfo.....	25	waitMS.....	8
optimizeHops.....	46	waitNewTick.....	9
pulseLEDG.....	15	wasRouted.....	45
pulseLEDR.....	13	writeToRAM.....	19
pulsingLEDG.....	14		

Sales and Service

Corporate office:

MICRORISC s.r.o., Delnicka 222, 506 01 Jicin, Czech Republic, EU
Tel: +420 493 538 125, Fax: +420 493 538 126, www.microrisc.com

Partners and distribution:

please visit www.iqrf.org/partners

Quality management:

ISO 9001 : 2000 certified

Trademarks:

*The IQRF name and logo are registered trademarks of MICRORISC s.r.o.
PIC, SPI, Microchip, RFM and all other trademarks mentioned herein are property of their respective owners.*

Legal:

All information contained in this publication is intended through suggestion only and may be superseded by updates without prior notice. No representation or warranty is given and no liability is assumed by MICRORISC s.r.o. with respect to the accuracy or use of such information.

Without written permission it is not allowed to copy or reproduce this information, even partially.

No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

The IQRF products utilize several patents (CZ, EU, US)

Website	www.iqrf.org
E-mail	sales@iqrf.org
On-line support	http://iq-esupport.com



Simple way to smarter wireless solutions