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

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	`void` **`reset()`**
Parameters	–
Return value	–
Output values	• All user RAM is cleared. The only register which keeps the original us value is `userStatus` – see the IQRF OS User's guide. • `-TO = 0`
Preconditions	–
Remarks	• This is forced watchdog overflow reset. To identify this type of reset the `-TO` and other status flags are available in the `userReg0` just after boot. See IQRF OS User's guide [1] (Reset). • Another helping way to analyze reset causation is via the `userStatus` 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	`if (Error == 1)` `  reset();`

calibrateTimer

Function	Calibrate tick generator
Purpose	Precise timing
Syntax	`void` **`calibrateTimer()`**
Parameters	–
Return value	–
Output values	Tick duration is calibrated to eliminate RC oscillator inaccuracy.
Preconditions	–
Remarks	• 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	`if ((NextHour = 1) or (TempChange > 5)) // Calibrate once an hour` `  calibrateTimer(); // or after rapid temperature change`

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	`void` **`iqrfSleep()`**
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 shoud 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	`// 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`

Example2	`// 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`
	`{ ... }  //  ...`

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	`void` **`setRFsleep()`**
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	0.6 mA typ. is saved. RF sesponse 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	`setRFsleep();  // Put the RF circuitry in Sleep mode`

setRFready

Function	Wake RF circuitry up
Purpose	To wake RF circuitry up in advance for faster response. Easy and efficient power management.
Syntax	`void` **`setRFready()`**
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	–
Side effects	–
See also	`setRFsleep, iqrfSleep, getSupplyVoltage, checkRF, RFTXpacket, RFRXpacket`

Example	`setRFready();  // Wake the RF circuitry up from RF sleep in advance`
	`...  // at least 2 ms before RF operation`
	`RFTXpacket();  // for immediate reaction`

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	`void` **`debug()`**
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	• 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, `debug` operates only under the following conditions:
	  • 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 *Check Mode* 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 *Check Mode* is enabled.
	  • SPI need not be enabled by `enableSPI`

Remarks	Number of `debug()` instances is unlimited. The application is running until a `debug` 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 *Skip Debug* button. Then the application program continues running until another `debug` function is encouneterd and so on. See IQRF IDE Help and example E04-EEPROM [10].
Side effects	• `param1` to `param4` are not displayed • Watchdog is cleared while in Debug mode
See also	–

Example	`if (compareBufferINFO2RF(4))`
	`  W = 1;  // match`
	`else`
	`  W = 2;  // mismatch`
	`debug();  // Skip Debug 1 or 2 will be displayed here according the result`

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	• 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	`if (getSupplyVoltage()>7)`
	`  …  // Voltage > 2.95V`
	`else`
	`  …  // Low battery`

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: `Ta = ADRES × 75/256 – 50`. See example E08–TEMPERATURE [10].
Side effects	–
See also	–

Example1	`  // Temperature measurement`
	`uns16  temperature;`
	`getTemperature();  // Temperature measurement`
	`temperature.high8 = ADRESH&0x03;  // 10 b result is stored  in ADRESH`
	`temperature.low8  = ADRESL;  // and ADRESL`

Example2	`  // 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`

Active waiting

waitMS

Function	Wait specified number of miliseconds
Purpose	Time delay generation
Syntax	`void` **`waitMS(ms)`**
Parameters	`ms` - time to wait in miliseconds (1 - 255)
Return value	–
Output values	–
Preconditions	This function can be combined with `waitDelay, startCapture` and `captureTicks`.
Remarks	This is an active waiting (on OS foreground). No other operation runs on OS foreground during waiting. Time precission depends on internal RC oscillator. Thus, the delay can vary with temperature etc. See respective PIC datasheet [8].
Side effects	–
See also	`waitDelay`, `startDelay, startLongDelay`

Example	`waitMS(10);  // Delay 10 ms. Program stays here for the whole 10 ms period`
	`  ...  // and continues here just after the period elapsed.`

waitDelay

Function	Wait specified number of ticks
Purpose	Time delay generation
Syntax	`void` **`waitDelay(ticks)`**
Parameters	`ticks` – time to wait in 10 ms intervals (1 - 255)
Return value	–
Output values	–
Preconditions	This function can be combined with `waitMS`.
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 `startDelay and startLongDelay`. 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 waitMS is more precise due to latency.
See also	`waitMS`, `startDelay, startLongDelay`

Example	`  // 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.`

waitNewTick

Function	Wait for a new tick
Purpose	Timing synchronization of user operations
Syntax	`void` **`waitNewTick()`**
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	`waitMS`, `waitDelay`
Example	`waitNewTick();`

Timing on background

startCapture

Function	Reset and start the Capture timer
Purpose	Initialization of time measurement or delay generation
Syntax	`void` **`startCapture()`**
Parameters	–
Return value	–
Output values	–
Preconditions	This function can be combined with `waitMS`.
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 `bondRequest`, `bondNewNode` and `RFRXpacket`.
See also	`captureTicks`
Example	See `captureTicks`

captureTicks

Function	Get number of ticks counted from the last `startCapture` and `captureTicks` calling.
Purpose	Measurement of elapsed time.
Syntax	`void` **`captureTicks()`**
Parameters	–
Return value	–
Output value	• `param3`: ticks counted from the last `startCapture` (0 - 65535) • `param4`: ticks counted from the last `captureTicks` (0 - 65535)
Preconditions	• `startCapture` should be used at least once before. • To ensure correct operation the counter must not overflow. That is why `captureTicks` should be called max. ~655 s after last `startCapture` or `captureTicks` calling.
Remarks	See example E05–DELAYS [10]
Side effects	Functionality is affected by `bondRequest`, `bondNewNode` and `RFRXpacket`. Internal ticks are based on internal RC oscillator. Thus, the delay can vary with temperature etc. See respective PIC datasheet [8].
See also	`startCapture`

Example	`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`

startDelay

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

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 `waitMS` and `startDelay`.
Remarks	The Delay timer measures specified time period on OS background. The result is available via the `isDelay` function.
Side effects	This function does not work properly if the `waitDelay, startDelay` functions are active.
See also	`isDelay, startDelay, waitDelay`
Example	See `isDelay`

isDelay

Function	Information whether specified delay is still in progress
Purpose	Time measurement or delay generation
Syntax	`bit` **`isDelay()`**
Parameters	–
Return value	• `1: still in progress` • `0: elapsed`
Output values	–
Preconditions	`startDelay` or `startLongDelay` should be used before.
Remarks	• The (Long)Delay timer measures specified time period. The result is available via the `isDelay` function. • Tip: the `clrwdt instruction` should be used to avoid unintentional watchdog reset during the delay. • See example `E05-DELAYS` [10].
Side effects	–
See also	`startDelay, startLongDelay`

Example1	`// 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`

Example2	`// 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`

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	`void` **`setOnPulsingLED(ticks)`**
Parameters	`uns8 ticks`: 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	`setOffPulsingLED, pulsingLEDR, pulseLEDR, pulsingLEDG, pulseLEDG`
Example	`See setOffPulsingLED`

setOffPulsingLED

Function	LEDs Off time setting (red as well as green)
Purpose	Specification of the "Off" time for LEDs (for blinking)
Syntax	`void` **`setOffPulsingLED(ticks)`**
Parameters	`uns8 ticks`: 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	`setOnPulsingLED, pulsingLEDR, pulsingLEDG`

Example	`// Change blinking to 250 ms On / 750 ms Off`
	`setOnPulsingLED(25);  // 250 ms On`
	`setOffPulsingLED(75);  // 750 ms Off`

pulsingLEDR

Function	Red LED blinking
Purpose	Continuous red LED blinking on OS background
Syntax	`void` **`pulsingLEDR()`**
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 `stopLEDR`).
Side effects	–
See also	`setOnPulsingLED, setOffPulsingLED, stopLEDR, pulseLEDR`
Example1	`pulsingLEDR();  // continuous blinking on OS background`

Example1	`   // Blinking for 2 s`
	`pulsingLEDR();  // blinking for 2 s on OS background`
	`waitDelay(200);  // 2 s delay generated on foreground`
	`stopLEDR();  // Stop blinking`

pulseLEDR

Function	Single red LED flash
Purpose	Red LED flash on OS background
Syntax	`void` **`pulseLEDR()`**
Parameters	–
Return value	–
Output values	–
Preconditions	Flash time should be defined in advance by `setOnPulsingLED`. 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 _LEDR output (the pin the red LED is connected to) and corresponding control bit (TRISx.x - see `IQRF-memory.h` header file).
Side effects	• 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 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	`setOnPulsingLEDR, pulsingLEDR, stopLEDR`

Example	`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`

stopLEDR

Function	Red LED off, blinking stopped
Purpose	Stops the red LED activity on OS background
Syntax	`void` **`stopLEDR()`**
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	–
Side effects	• 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	`pulsingLEDR();  // Start blinking on OS background`
	`  ...  // Blinking continues during any operation`
	`stopLEDR();  // Stop blinking`

Example2	`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`

pulsingLEDG

Function	Green LED blinking
Purpose	Continuous green LED blinking on OS background
Syntax	`void` **`pulsingLEDG()`**
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	`pulsingLEDG();  // continuous blinking on OS background`

Example1	`  // Blinking for 2 s`
	`pulsingLEDG();  // blinking for 2 s on OS background`
	`waitDelay(200);  // 2 s delay generated on foreground`
	`stopLEDG();  // Stop blinking`

pulseLEDG

Function	Single green LED flash
Purpose	Green LED flash on OS background
Syntax	`void` **`pulseLEDG()`**
Parameters	–
Return value	–
Output values	–
Preconditions	Flash time should be defined in advance by `setOnPulsingLED`. 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 _LEDG output (the pin the green LED is connected to) and corresponding control bit (TRISx.x - see `IQRF-memory.h` header file).
Side effects	• The appropriate PIC pin is not restored to the state before pulsingLEDG/pulseLEDG (TRISx.x == 0, _LEDG == 0 after finishing on background). • Background `activity` overrides setting of _LEDG (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	`setOnPulsingLEDG, pulsingLEDG, stopLEDG`

Example	`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`

stopLEDG

Function	Green LED off, blinking stopped
Purpose	Stops the green LED activity on OS background
Syntax	`void` **`stopLEDG()`**
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	–
Side effects	• The appropriate PIC pin is not restored to the state before `pulsingLEDG`/`pulseLEDG` (TRISx.x == 0, _LEDG == 0 after finishing on background). • Background `activity` overrides setting of _LEDG (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	`pulsingLEDG, pulseLEDG`

Example1	`pulsingLEDG();  // Start blinking on OS background`
	`  ...  // Blinking continues during any operation`
	`stopLEDG();  // Stop blinking`

Example2	`pulseLEDG();  // Green LED On on OS background`
	`  ...  // continuously lighting during any operation`
	`  // until specified time expired`
	`stopLEDG();  // or LED is switched Off by this command`

EEPROM

eeReadByte

Function	Read one byte from specified location in EEPROM
Purpose	Access to EEPROM
Syntax	`uns8` **`eeReadByte(addr)`**
Parameters	`uns8 addr`: 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	• Direct user `access` to EEPROM (using registers `EECONx` 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	`eeReadData, eeWriteByte, eeWriteData`
Example1	`i = eeReadByte(0);  // store 1 byte from EEPROM from address 0 to i`
Example2	`// Illegal access: Avoid access to EEPROM locations 192 (0xC0) or higher` `i = eeReadByte(200);  // Reading from protected area is redirected to 160 (0xA0)`

eeReadData

Function	Read a block of specified length from specified location in EEPROM to `bufferINFO`
Purpose	Block access to EEPROM
Syntax	`void` **`eeReadData(addr, length)`**
Parameters	• `uns8 addr`: address in EEPROM (`0` to `0xBF - length + 1`). See EEPROM map [2]. • `uns8 length`: number of bytes to be read (1 to 32)
Return value	–
Output values	`bufferINFO[0` to `length - 1]`
Preconditions	–
Remarks	• Direct user `access` to EEPROM (using registers `EECONx` 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	`eeReadByte, eeWriteByte, eeWriteData`

Example1	`eeReadData(10, 16);  // copy 16B from EEPROM from address 10 to bufferINFO`
	`//  bufferINFO[0] = EEPROM[10]`
	`//  ...`
	`//  bufferINFO[15] = EEPROM[25]`

Example2	`// 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]`

eeWriteByte

Function	Write one byte to specified location in EEPROM
Purpose	Access to EEPROM
Syntax	`void` **`eeWriteByte(addr, data)`**
Parameters	• `uns8 addr`: address in EEPROM (0xA0 to 0xBF for Coordinator and 0 to 0xBF for other devices). See EEPROM map [2]. • `uns8 data`: value to be written (0 to 255)
Return value	–
Output values	–
Preconditions	–
Remarks	• Direct user access to EEPROM (using registers `EECONx` 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	`eeReadByte, eeReadData, eeWriteData`
Example1	`eeWriteByte(191, 0x75)  // store 0x75 to EEPROM to address 191` `eeWriteByte(0x80, X)    // copy X to EEPROM to address 0x80`
Example2	`  // Illegal access: Avoid access to EEPROM locations 192 (0xC0) or higher` `eeWriteByte(198, 0x75);  // Attempt to write to protected area – nothing is written.`

eeWriteData

Function	Write a block of specified length from `bufferINFO` to specified location in EEPROM
Purpose	Block access to EEPROM
Syntax	`void` **`eeWriteData(addr, length)`**

Parameters	• `uns8 addr`: address in EEPROM . See EEPROM map [2].
	  • (`0xA0` to `0xBF – length + 1`) for Coordinator
	  • (`0` to `0xBF – length + 1`) for other devices
	• `uns8 length`: number of bytes to be written from `bufferINFO` (1 to 32)

Return value	–
Output values	–
Preconditions	–
Remarks	• Direct user access to EEPROM (using registers `EECONx` 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	`eeReadByte, eeReadData, eeWriteByte`

Example1	`eeWriteData(10,16);  // copy 16B from bufferINFO to EEPROM to address 10`
	`//  EEPROM[10] = bufferINFO[0]`
	`//  ...`
	`//  EEPROM[25] = bufferINFO[15]`

Example2	`  // Illegal access: Avoid access to EEPROM locations 192 (0xC0) or higher`
	`eeWriteData(200,16);  // Attempt to write to protected area – nothing is`
	`// written.`

RAM

readFromRAM

Function	Read one byte from specified location in RAM
Purpose	Indirect access to RAM registers
Syntax	`uns8` **`readFromRAM(addr)`**
Parameters	`uns8 addr`: memory location address. "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.
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 `X = Y;`) 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 `readFromRAM` and `writeToRAM`. See example E06–RAM [10].
Side effects	–
See also	`writeToRAM`

Example1	`IRP = 0;`
	`X = readFromRAM(bufferCOM + 10);`
	`IRP = 1;`
	`Y = readFromRAM(bufferRF + 10);`

Example2	`// Not correct:`
	`IRP = 0;  // To access 0x190 IRP must be set to 1`
	`X = readFromRAM(0x190) - 1;  // 0x90 is addressed instead of expected 0x190`

Example3	`// Correct:`
	`IRP = 1;`
	`X = readFromRAM(0x90) - 1;  // Perfect`
	`X = readFromRAM(0x190) - 1;  // Also works thanks to the compiler feature`

Example4	`// Not allowed. The compiler uses FSR in such cases.`
	`for (i=0; i<5; i++)`
	`{`
	`  A = bufferRF[i];`
	`  …`
	`}`

Example5	`// Correct`
	`for (i=0; i<5; i++)`
	`{`
	`  A = readFromRAM(bufferRF + i);`
	`  …`
	`}`

writeToRAM

Function	Write one byte to specified location in RAM
Purpose	Indirect access to RAM registers
Syntax	`void` **`writeToRAM(addr, value)`**
Parameters	• uns8 addr: memory location address. Unlike `copyMemoryBlock`, "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. • `uns8 value`: value to be written
Return value	–
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. • 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 `X = Y;`) 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 `readFromRAM` and `writeToRAM`. See example E06–RAM [10]. Certain RAM locations are not accessible at all for security reasons.
Side effects	–
See also	`readFromRAM, copyMemoryBlock`

Example1	`IRP = 0;`
	`writeToRAM(bufferCOM + 10, 2 * X);`
	`IRP = 1;`
	`writeToRAM(bufferRF + 10, 2 * X);`

Example2	`// Not correct:`
	`IRP = 0; // To access 0x190 IRP must be set to 1`
	`writeToRAM(0x190, 201); // 0x90 is addressed instead of expected 0x190`

Example3	`// Correct:`
	`IRP = 1;`
	`writeToRAM(0x90, 201); // Perfect`
	`writeToRAM(0x190, 201); // Also works thanks to the compiler feature`

Example4	`// Not allowed. The compiler uses FSR in such cases.`
	`for (i=0; i<5; i++)`
	`bufferRF[i] = i;`

Example5	`// Correct`
	`for (i=0; i<5; i++)`
	`writeToRAM(bufferRF + i, i);`

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 bufferINFO to bufferCOM
Purpose	Data transfer between buffers
Syntax	void **copyBufferINFO2COM()**
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets memoryOffsetFrom and memoryOffsetTo are applied (see above).
Remarks	• See example E06 - RAM [10]. • If memoryOffsetFrom = 0 and memoryOffsetTo = 0 complete 35 B is copied.
Side effects	–
See also	clearBufferINFO, copyBufferINFO2RF, copyBufferRF2COM, copyBufferRF2INFO, copyBufferCOM2RF, copyBufferCOM2INFO, compareBufferINFO2RF, copyMemoryBlock
Example1	`copyBufferINFO2COM();`

Example2	`memoryOffsetFrom = 0;  // bufferINFO to be copied`
	`memoryOffsetTo = 10;  // to  bufferCOM starting from bufferCOM[10].`
	`copyBufferINFO2COM;  // Just first 25 B is copied (until bufferCOM full).`

copyBufferINFO2RF

Function	Copy bufferINFO to bufferRF
Purpose	Data transfer between buffers
Syntax	void **copyBufferINFO2RF()**
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets memoryOffsetFrom and memoryOffsetTo are applied (see above).
Remarks	• See example E06 - RAM [10]. • If memoryOffsetFrom = 0 and memoryOffsetTo = 0 complete 35 B is copied.
Side effects	–
See also	clearBufferINFO, copyBufferINFO2COM, copyBufferRF2COM, copyBufferRF2INFO, copyBufferCOM2RF, copyBufferCOM2INFO, compareBufferINFO2RF, copyMemoryBlock
Example	`copyBufferINFO2RF();`

copyBufferRF2COM

Function	Copy `bufferRF` to `bufferCOM`
Purpose	Data transfer between buffers
Syntax	`void` **`copyBufferRF2COM()`**
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets `memoryOffsetFrom` and `memoryOffsetTo` are applied (see above).
Remarks	• See example E06 - RAM [10]. • If memoryOffsetFrom = 0 and memoryOffsetTo = 0 complete 35 B is copied.
Side effects	–
See also	`clearBufferINFO, copyBufferINFO2RF, copyBufferINFO2COM, copyBufferRF2INFO, copyBufferCOM2RF, copyBufferCOM2INFO, compareBufferINFO2RF, copyMemoryBlock`
Example	`copyBufferRF2COM();`

copyBufferRF2INFO

Function	Copy `bufferRF` to `bufferINFO`
Purpose	Data transfer between buffers
Syntax	`void` **`copyBufferRF2INFO()`**
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets `memoryOffsetFrom` and `memoryOffsetTo` are applied (see above).
Remarks	• See example E06 - RAM [10]. • If memoryOffsetFrom = 0 and memoryOffsetTo = 0 complete 35 B is copied.
Side effects	–
See also	`clearBufferINFO, copyBufferINFO2COM, copyBufferINFO2RF, copyBufferRF2COM, copyBufferCOM2RF, copyBufferCOM2INFO, compareBufferINFO2RF, copyMemoryBlock`
Example	`copyBufferRF2INFO();`

copyBufferCOM2RF

Function	Copy `bufferCOM` to `bufferRF`
Purpose	Data transfer between buffers
Syntax	`void` **`copyBufferCOM2RF()`**
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets `memoryOffsetFrom` and `memoryOffsetTo` are applied (see above).
Remarks	• See example E06 - RAM [10]. • If memoryOffsetFrom = 0 and memoryOffsetTo = 0 complete 35 B is copied.
Side effects	–
See also	`clearBufferINFO, copyBufferINFO2COM, copyBufferINFO2RF, copyBufferRF2COM, copyBufferRF2INFO, copyBufferCOM2INFO, compareBufferINFO2RF, copyMemoryBlock`
Example	`copyBufferCOM2RF();`

copyBufferCOM2INFO

Function	Copy `bufferCOM` to `bufferINFO`
Purpose	Data transfer between buffers
Syntax	`void` **`copyBufferCOM2INFO()`**
Parameters	–
Return value	–
Output values	–
Preconditions	Offsets `memoryOffsetFrom` and `memoryOffsetTo` are applied (see above).
Remarks	• See example E06 - RAM [10]. • If memoryOffsetFrom = 0 and memoryOffsetTo = 0 complete 35 B is copied.
Side effects	–
See also	`clearBufferINFO, copyBufferINFO2COM, copyBufferINFO2RF, copyBufferRF2COM, copyBufferRF2INFO, copyBufferCOM2RF, copyMemoryBlock`
Example	`copyBufferCOM2INFO();`

compareBufferINFO2RF

Function	Compare `bufferINFO` and `bufferRF` with respect to specified length
Purpose	Buffer comparison
Syntax	`bit` **`compareBufferINFO2RF(length)`**
Parameters	`uns8 length`: number of bytes to be compared (1 to 35)
Return value	• 1 – match • 0 – mismatch
Output values	–
Preconditions	–
Remarks	• See example E06 - RAM [10]. • If memoryOffsetFrom = 0 and memoryOffsetTo = 0 complete 35 B is copied.
Side effects	–
See also	`clearBufferINFO, copyBufferINFO2RF, copyBufferRF2INFO, swapBufferINFO`
Example	`if (!compareBufferINFO2RF(32)) // Compare 32 B` `  then Error = 1; // Error if mismatch`

copyMemoryBlock

Function	Copy specified RAM block to specified location
Purpose	Copy memory block within RAM
Syntax	`void` **`copyMemoryBlock`** `(from, to, length)`
Parameters	• `uns16 from`: starting address of the block to be copied • `uns16 to`: destination address • `uns8 length`: block length in bytes
Return value	–
Output values	–
Preconditions	• 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 `writeToRAM`, "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	• *This function can access RAM within the whole memory area.* • 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	`writeToRAM`

Example	`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]`

swapBufferINFO

Function	Swap `bufferINFO` and `bufferAUX`
Purpose	Temporary `bufferINFO` saving
Syntax	`void` **`swapBufferINFO()`**
Parameters	–
Return value	–
Output values	Content of `bufferINFO` and `bufferAUX` is swapped. See example E06 - RAM [10].
Preconditions	–
Remarks	32 B are swapped
Side effects	–
See also	`moduleInfo, appInfo`

Example	`swapBufferInfo();  // Temporarily save bufferInfo to bufferAUX`
	`appInfo();  // Get user data from EEPROM`
	`...`
	`swapBufferInfo();  // and restore previous data in bufferInfo`

clearBufferINFO

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

clearBufferRF

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

moduleInfo

Function	Store Module data to `bufferINFO`
Purpose	Get information about transceiver module and OS
Syntax	`void` **`moduleInfo()`**
Parameters	–
Return value	–
Output values	`bufferINFO[0 to 7]`
Preconditions	–
Remarks	<table><tr><td>address in `bufferInfo`</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td rowspan="2">meaning</td><td colspan="2" rowspan="2">OS build</td><td rowspan="2">PIC type</td><td rowspan="2">OS version</td><td>Coordinator / Node</td><td colspan="3">serial number</td></tr><tr><td colspan="4">Module ID</td></tr></table> See IQRF OS User's guide [1] (Identification) for Module data format details.
Side effects	–
See also	`appInfo`

Example	`uns24 SN @ bufferInfo;`
	`uns8 OSv  @ bufferInfo[4];`
	`moduleInfo();  // Now SN == module serial number`
	`// and OSv == OS version`

appInfo

Function	Store Application data from EEPROM to `bufferINFO`
Purpose	Get information about user application
Syntax	`void` **`appInfo()`**
Parameters	–
Return value	–
Output values	`bufferINFO[0 to 31]`
Preconditions	–
Remarks	See IQRF OS User's guide [1] (Identification and Appendix, Memory maps).
Side effects	–
See also	`moduleInfo`

Example1	`appInfo();  // Copy Application data from EEPROM to bufferINFO`
	`copyBufferINFO2RF();  // and then to bufferRF`

Example2	`#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`

SPI

enableSPI

Function	Activate SPI communication module and related pins
Purpose	Enable SPI communication
Syntax	`void` **`enableSPI()`**
Parameters	–
Return value	–
Output values	SPI Status is switched to *SPI ready, communication mode*.
Preconditions	–
Remarks	• The PIC internal SPI hardware module and appropriate pins (C5 – C8) are cofigured 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 `disableSPI`.
See also	`disableSPI, startSPI, stopSPI, getStatusSPI, restartSPI`
Example	`See getStatusSPI`

disableSPI

Function	Switch SPI HW module off and configure SPI pins as I/Os
Purpose	Disable SPI communication
Syntax	`void` **`disableSPI()`**
Parameters	–
Return value	–
Output values	SPI Status is switched to *SPI not active*.
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	• The appropriate PIC pins are not restored to the state before enableSPI calling. • Current packet is lost by both sides if SPI communication is running on background at this moment.
See also	`enableSPI, startSPI, stopSPI, getStatusSPI, restartSPI`
Example	`See getStatusSPI`

startSPI

Function	Indicate ready to Master.
Purpose	• Initiate SPI packet transmission from Slave (request to Master). Provide data from `bufferCOM` to Master according to Master's clock (on OS background). • `startSPI(0)` indicates to Master that the Slave is ready to receive data (`bufferCOM` not full).
Syntax	`void` **`startSPI(length)`**
Parameters	`uns8 length`: number of bytes to be sent (0 to 41)
Return value	–
Output values	SPI Status is switched to: • *SPI data ready* – after `startSPI(1` to `41)` • *SPI ready, Communication mode* – after `startSPI(0).`
Preconditions	SPI must be enabled by the `enableSPI` function before.
Remarks	• SPI runs on OS background. • `startSPI(0)` 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	`enableSPI, disableSPI, stopSPI, getStatusSPI, restartSPI`

Example1	`// Slave -> Master`
	`bufferCOM[0] = "I";`
	`bufferCOM[1] = "Q";`
	`enableSPI();`
	`startSPI(2);  // Request to Master is active on backgroung from now`
	`...  // and the program just continues here`

Example2	`startSPI(0);  // Reset SPI communication`
Example3	`See getStatusSPI`

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 *User stop*.
Preconditions	–
Remarks	• `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	`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`
	`}`

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 preceeding `stopSPI`.
Remarks	SPI can continue from the state just before `stopSPI`.
Side effects	–
See also	`startSPI, stopSPI`

Example1	`startSPI(16);  // SPI started`
	`  ...`
	`stopSPI();  // SPI stopped temporarily`
	`  ...  // to make some operations`
	`restartSPI();  // and allow to continue`

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` **`getStatusSPI()`**
Parameters	–
Return value	• 1 – SPI busy • 0 – SPI not busy
Output values	• `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 `enableSPI`
Remarks	• 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	`enableSPI, disableSPI, startSPI, stopSPI, restartSPI`

Example1

```
     // 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.
```

Example2

```
enableSPI();
startSPI(2);                // 2 B to send to master
while (getStatusSPI())      // Wait until SPI is not busy
  waitMS(1);
...                         // Now the transfer is finished
```

RF

setTXpower

Function	Set RF output power
Purpose	Change RF range
Syntax	`void` **`setTXpower(level)`**
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	`setTXpower(7);  // Max. RF output power`

setRFspeed

Function	Select RF bit rate
Purpose	Select RF bit rate
Syntax	**`void setRFspeed(speed)`**
Parameters	`uns8 speed:` • 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	• 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	`setRFspeed(1);  // 1.2 kb/s selected` `setRFchannel(...);  // channel must be selected then`
Example2	`setRFspeed(2);  // 19.2 kb/s selected` `setRFchannel(...);  // channel must be selected then`

setRFband

Function	Select RF frequency band
Purpose	Select 868 MHz or 916 MHz band
Syntax	**`void setRFband(band)`**
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	`setRFband(1);  // 916 MHz band selected`
Example2	`setRFband(0);  // 868 MHz band selected`

setRFchannel

Function	Set RF channel
Purpose	Select free RF channel for not interfered communication
Syntax	**`void setRFchannel(channel)`**
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	`setRFband(0);  // 868 MHz band selected`
	`setRFspeed(3);  // 57.6 kb/s bit rate selected`
	`setRFchannel(25);  // 868.15 MHz channel selected`

setRFmode

Function	Set RF mode
Purpose	Specify power management and signal filtering modes for RF transmission and receipt
Syntax	**`void setRFmode(mode)`**
Parameters	`uns8 mode: SWTTFFRR` in binary • S: **Stay in RX mode** (for fast response for following `checkRF`, `RFRXpacket` or `RFTXpacket`) 1 RX chain stays enabled after `RFRXpacket` and `RFTXpacket` 0 RX chain is disabled after `RFRXpacket` and `RFTXpacket` • W: **Wait packet end** 1 Waits until receipt is finished if it is actually started even though toutRF timeout is over meanwhile. 0 `RFRXpacket` is unconditionally finished when toutRF timeout is over. • TT: **TX mode** 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 (RR ≠ 0). Signal with lower level is ignored. Relative RF range is shortened due to this filtration. The level corresponds to the `checkRF(x)` parameter: 00 x = 5 01 x = 20 10 x = 35 11 x = 50 • RR: **RX mode** 00 STD RX mode (Standard, transmitting device should have TT=00) 01 LP RX mode (Low power, transmitting device should have TT=01) 10 XLP RX mode (Extra low power, transmitting device should have TT=10) 11 RFIM mode (`RFRXpacket` is terminated when signal strength falls below the FF 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 `setRFmode` after finishing background tasks. See Example 2.
See also	`checkRF`

Example1

```
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)
```

Example2

```
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
```

checkRF

Function	Check incoming RF signal strength for specified level.
Purpose	Incoming RF signal detection to start RF receiving.
Syntax	**`bit checkRF(level)`**

Parameters	`uns8 level = DQI + RSSI_FILTER`
	• `DQI` (Data Quality Indicator, see the RF IC datasheet):
	  • `0x80` DQI enabled
	  • `0` DQI disabled
	• `RSSI_FILTER`: 0 to 64
	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.

Return value	• `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 ADRESH (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	• Higher level means lower sensitivity which requires stronger signal resulting in higher immunity against interefrences 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 S = 1 in `setRFmode`) • Checking consumes ~9.5 mA. • This function is intended for fast response and power consumption reduction in STD RX mode.
Side effects	–
See also	`setRFmode, RFRXpacket`

Example1	`// 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`

Example2	`if (checkRF(10))  // Detect signal with RSSI > 42` `  ...`

Example3	`// RF signal strength analyzer`
	`SWDTEN = 0;  // disable watchdog`
	`while (1)`
	`  if (checkRF(3)) pulseLEDR(); // LED flash if signal level >= 3 detected`

RFTXpacket

Function	Send RF packet of specified length from `bufferRF`.
Purpose	RF transmission
Syntax	`void` **`RFTXpacket()`**
Parameters	–
Return value	–
Output values	–

Preconditions	• Peer-to-peer topology:
	  • `PIN = 0` (Peer-to-peer)
	  • `DLEN =` packet length in bytes (0 to 64)
	  • Prepare data to send in `bufferRF[0] to bufferRF[DLEN - 1]` (if `DLEN ≠ 0`)
	  • Set RF output power via `setTXpower`
	• IQMESH:
	  • `PIN = 0x80` (IQMESH)
	  • Other network related parameters should also be specified
	See IQRF OS User's guide [1] and IQMESH specification [4].

Remarks	• 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	• bufferRF[DLEN] and bufferRF[DLEN+1] are destroyed • The RF circuitry wakes up (in case of sleeping).
See also	`RFRXpacket, setTXpower, setRFmode` and (in case of IQMESH) also other RF functions

Example1

```
   // 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
```

Example2

```
   // 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)
```

Example3

```
   // 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)
```

RFRXpacket

Function	Receive RF packet to `bufferRF` and provide related information
Purpose	RF receiving
Syntax	`bit` **`RFRXpacket()`**
Parameters	–
Return value	• 1 – packet received • 0 – packet not received

Output values	• lastRSSI – the RSSI value after successful receipt (single sample). Quiet level (noise) is 25 - 28.
	• `DLEN =` packet length. This variable is destroyed if the receipt is not successful.
	• PIN is updated according to packet received. This variable is destroyed if the receipt is not successful.
	• `_NTWPACKET`: valid if `RFRXpacket` return value == 1 only:
	  • 1 – networking packet received
	  • 0 – non-networking packet received
	• Other related networking information in case of IQMESH.

Preconditions	• Timeout should be specified in `toutRF` (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	• 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 `setRFmode`. • If the packet is sent when the addresse (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 `setNetworkFilteringOn` and `setNetworkFilteringOff`. • 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	• Update PIN before every `RFTXpacket` folowed after `RFRXpacket`. • Result of `captureTicks` is destroyed if `startCapture` is active on background at the same time. • bufferRF[DLEN] and bufferRF[DLEN+1] 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	`RFTXpacket, setRFmode, checkRF` and (in case of IQMESH) also other RF functions

Example 1

```
   // 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.
}
```

Example 2

IQMESH: See `setNodeMode` and `setNetworkFilteringOn`.

Example 3

```
if (RFRXpacket())
{
  if (_ROUTEF)             // Was the packet routed?
  {                        // Yes - wait for finish of routing
    while (RTDT0)          // RTDT0 - rest of hops
    {
      waitDelay(RTDT1);    // RTDT1 - timeslot
      RTDT0--;             // Do not answer until all hops are finished
    }
  }
  ...                      // Now the Node is allowed to answer
}
```

Networking

setCoordinatorMode

Function	Set Coordinator mode
Purpose	Assign the TR module as a network Coordinator
Syntax	`void` **`setCoordinatorMode()`**
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 `RFRXpacket` and `RFTXpacket`.
Side effects	–
See also	`setNodeMode, setNonetMode`
Example	

setNodeMode

Function	Set Node mode
Purpose	Assign the TR module as a network Node
Syntax	`void` **`setNodeMode()`**
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 `RFRXpacket` and `RFTXpacket`.
Side effects	–
See also	`setCoordinatorMode, setNonetMode`
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	• Default OS mode is Peer-to-peer. • This settings affects `RFRXpacket` and `RFTXpacket` features. • PIN is not affected immediately but it is cleared after subsequent `RFRXpacket` or `RFTXpacket`.
Side effects	–
See also	`setCoordinatorMode, setNodeMode`

Example	`setNetworkOne();  // TR communicates in IQMESH networking mode here`
	`  ...  //`
	`setNonetMode();  // Switch to Peer-to-peer mode`
	`  ...  // Now TR communicates without networking support`

setNetworkFilteringOn

Function	Start filtering incoming non-networking packets and packets coming from non-current network.
Purpose	To receive packets from current network only.
Syntax	`void` **`setNetworkFilteringOn()`**
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	`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.`

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	`void` **`setNetworkFilteringOff()`**
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	`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`

setUserAddress

Function	Assign a user address to a Node
Purpose	User addressing of Nodes
Syntax	`void` **`setUserAddress(address)`**
Parameters	`uns16 address`: user address 1 to 65 000
Return value	–
Output values	–
Preconditions	For IQMESH Node and DFM2B only.
Remarks	• 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 procedureby 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	`setUserAddress(2000);  // The Node has got user address 2000`

Example 2	`setUserAddress(2000);`
	`  ...`
	`reset();  // User address is lost after reset`
	`setUserAddress(2000);  // User address restored`

Example 3	`setUserAddress(UA);`
	`eeWriteByte(EEUA, UA)  // User address stored to EEPROM`
	`  ...`
	`reset();  // User address lost after reset`
	`setUserAddress(eeReadByte(EEUA));  // User address restored from EEPROM`

getNetworkParams

Function	Get network parameters
Purpose	Get information about the network
Syntax	`uns8` **`getNetworkParams()`**
Parameters	–
Return value	<table><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><tr><td>networkTwo</td><td>networkingMode</td><td>routingOn</td><td>CoordinatorMode</td><td>AUXB</td><td>–</td><td>–</td><td>–</td></tr></table> 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
Output values	• `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	`if (amIBonded())  // Is the Node bonded?`
	`{  // Yes:`
	`  getNetworkParams();  //  Get Node number`
	`  myAddr = param2;`
	`}`

Routing

setRoutingOn

Function	Routing enabled
Purpose	Allow the Node to route packets on background.
Syntax	`void` **`setRoutingOn()`**
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH Nodes only. Default OS condition is Routing on.
Remarks	• 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	`setRoutingOff, discovery, isDiscovederedNode, wasRouted`
Example	–

setRoutingOff

Function	Routing disabled
Purpose	Forbid the Node to route packets on background.
Syntax	`void` **`setRoutingOff()`**
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	`setRoutingOn, discovery, isDiscovederedNode, wasRouted`
Example	–

discovery

Function	Discover Nodes for routing and asign VRN (Virtual Routing Number) to individual Nodes
Purpose	Routing backbone creation (for routing transparent from the user's point of view)
Syntax	`uns8` **`discovery(zones)`**
Parameters	`uns8 zones`: max. number of zones to be established (= max. number of hops allowed per packet)
Return value	Number of discovered Nodes (≤ number of bonded Nodes)
Output values	• Routing backbone is stored in EEPROM
Preconditions	• For IQMESH Coordinator only. • Nodes must be in the `answerSystemPacket` loop routine during Discovery.
Remarks	• 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 `discovery` 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	• Watchdog is disabled during this operation and enabled after finishing. • All OS buffers (`bufferINFO`, `bufferCOM`, `bufferRF` and `bufferAUX`) are destroyed
See also	`setRoutingOn`, `setRoutingOff`, `isDiscovederedNode`, `bondNewNode`, `answerSystemPacket`

Example1	`setTXpower(DISCOVERY_POWER);  // Set RF power for discovery`
	`nodes = discovery(10);  // Limit to max. 10 hops`
	`SWDTEN = 0;  // Possibly restore WDT`

Example2	`nodes = discovery(eeReadByte(0x00));  // Limit to number of bonded Nodes`
	`//  e.g. for Chain MESH`

answerSystemPacket

Function	Enable response to Coordinator for Discovery
Purpose	Discovery support from the Node's side
Syntax	`void` **`answerSystemPacket()`**
Parameters	–
Return value	–
Output values	Routing information exchanged between Coordinator and the Node via system packets.
Preconditions	• For IQMESH Node only. • Nodes must be in the `answerSystemPacket` loop routine when Discovery is running. • WDT should be disabled before `answerSystemPacket`
Remarks	• Nodes use the TX output power currently set in Coordinator for discovery. It is recommended to run `discovery` 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	• `toutRF` is changed after Discovery • RF output power is inherited from the Coordinator and not restored
See also	`setRoutingOn`, `setRoutingOff`, `isDiscovederedNode`, `discovery`
Example	

```
toutRF= MY_TOUT_RF;
if (RFRXpacket())
{
  …
}
else
{
  if (lastRSSI > discovery_threshold)
  {
    SWDTEN = 0;
    answerSystemPacket();   // to be discovered
    SWDTEN = 1;
    setTXpower(MY_POWER);
  }
}
```

isDiscoveredNode

Function	Check for being discovered
Purpose	Ask whether the Node has been discovered
Syntax	`bit` **`isDiscoveredNode(address)`**
Parameters	`uns8 address`: Node address
Return value	• `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	`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`
	`  …`
	`}`

wasRouted

Function	Indicate incoming packet routing
Purpose	To distinguish whether incoming packet has been routed for other recipient(s).
Syntax	`bit` **`wasRouted()`**
Parameters	–
Return value	• `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, isDiscovederedNode`

Example	`if (RFRXpacket())`
	`{`
	`  if (wasRouted())`
	`  pulseLEDG();  // indicate routing received packet for broadcast`
	`  …`
	`}`
	`else`
	`{`
	`  if (wasRouted())`
	`  pulseLEDG();  // indicate routing incoming packet for another addressee`
	`}`

optimizeHops

Function	Optimize number of hops for given Node
Purpose	Set optimized number of hops according to a topology, without flooding
Syntax	`void` **`optimizeHops(x)`**
Parameters	`uns8 x`: optimizing method • 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. *Not implemented yet.*
Return value	–
Output values	`RTDT0` (number of hops) is set
Preconditions	• 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, isDiscovederedNode`

Example	`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();`

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` **`bondRequest()`**
Parameters	–
Return value	• 1 – Node has been bonded • 0 – Node has not been bonded
Output values	• The `amIBonded` function starts to return value == 1 whenever is called while the Node is bonded by `bondRequest` not beeing unbonded by `removeBond` or `wipeBondNR`. • Coordinator is not affected at all. • `param2`: Node address (if successfully bonded only). Not guarranted for future OS versions.
Preconditions	For IQMESH only.
Remarks	• 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	The following values are modified and not restored: • DLEN • PIN • toutRF • bufferRF[0 to 63] destroyed • bufferINFO[0 to 34] destroyed • Result of `captureTicks` is destroyed if `startCapture` is active on background at the same time. • Watchdog is disabled during this operation and enabled after finishing • IQMESH mode must be restored by `setNodeMode` after `bondRequest`
See also	`bondNewNode, amIBonded, removeBond, rebondNode, getNetworkParams, setNodeMode`

Example1	`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`
	`}`
	`setNodeMode(); // Restore`
	`stopLED();`

Example2	See `amIBonded`

amIBonded

Function	Is the Node bonded?
Purpose	Test whether the Node is bonded on Node's side
Syntax	`bit` **`amIBonded()`**
Parameters	–

Return value	• 1 – Node is bonded (after `bondRequest` not beeing unbonded by `removeBond`)
	• 0 – Node is not bonded:
	  • no `bondRequest` has ever been successfully executed
	  • after `removeBond`

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	`bondRequest, removeBond`

Example	`while (!amIBonded())  // Request for beeing bonded (if not bonded yet)`
	`{`
	`  bondRequest();  // Repeatedly try to bond`
	`  clrwdt();`
	`}  // until successful`

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` **`removeBond()`**
Parameters	–
Return value	–
Output values	• The amIBonded `function` starts to return value == 0 whenever is called until the Node is bonded again via `bondRequest`. • 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	• See example E11 - IQMESH-N [10]. • For rebonding use bondRequest again. • removeBond relates to Node only and removeBondedNode and rebondNode 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	• The effect of `removeBond` will not surfice in Coordinator until `bondRequest/bondNewNode` procedure is invoked afterward. That is why Coordinator can use the number reserved for this Node for another Node meanwhile (after possible `removeBondedNode`). In this case the Coordinator will assign a different number.
See also	`bondRequest, bondNewNode, amIBonded, rebondNode`
Example	`removeBond();  // Remove the bond.`

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	`uns8 address` • 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 `setUserAddress`. It is intended especially for networks with more than 239 Nodes.
Return value	• 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	• `param2`: Node number • `bufferRF[0 to 1]`: two lower ID bytes of the Node (is successfully bonded), LSB in `bufferRF[0]`. • The `isBondedNode` function starts to return value == 1 whenever is called while the Node is in the list of bonded Nodes.
Preconditions	• For IQMESH Coordinator only. • Coordinator accomplishes bonding on request from Node via RF. When this function is executing the `bondRequest` function must just be active in the Node.
Remarks	• 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 `removeBondedNode(address)`. • No stack level is available to call this function in subroutine.
Side effects	The following values are modified and not restored: • toutRF modified • bufferRF[0 to 63] = destroyed • bufferINFO[0 to 34] = destroyed • Result of `captureTicks` is destroyed if `startCapture` is active on background at the same time. • Watchdog is disabled during this operation and enabled after finishing • IQMESH mode must be restored by `setCoordinatorMode`
See also	`bondRequest, removeBondedNode, rebondNode, isBondedNode, setUserAddress, setCoordinatorMode`

Example	`if (bondNewNode()) // Bonding successful ?`
	`{ // Yes:`
	`  NodeNumber = param2;`
	`  ...`
	`}`
	`else`
	`{ // No:`
	`  ... // Arrange necessary steps`
	`}`
	`setCoordinatorMode(); // Restore`

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	• 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	`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`
	`}`

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 `isBondedNode` 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	`removeBondedNode` and `rebondNode` relate to Coordinator only and `removeBond` 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	`removeBondedNode(28);  // Coordinator assumes Node #28 to be`
	`// out of the network from now on`

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	`bit` **`rebondNode(n)`**
Parameters	`uns8 n`: Node number
Return value	reserved for future OS versions
Output values	The `isBondedNode` 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 beeing bonded ever before.
Remarks	`removeBondedNode` and `rebondNode` relate to Coordinator only and `removeBond` 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`, `removeBondedNode`, `isBondedNode`

Example	`rebondNode(28);  // Coordinator assumes Node #28 to be`
	`// back in the network from now on`

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	`void` **`clearAllBonds()`**
Parameters	–
Return value	–
Output values	The `isBondedNode` 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	bufferINFO[0 to 34] destroyed
See also	`removeBondedNode`
Example	`clearAllBonds();  // Exclude all currently bonded nodes from the network`

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

- 110124 Example for RFRXpacket and side effect for `bondRequest` and `bondNewNode` added.
- 110112 Information added and precised.
- 101223 OS v3.0, first release for TR-52B, TR-53B and compatibles.

Index

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