IQRF OS

Operating System

version 2.10
for TR-31B and TR-32B

Reference Guide

Simple way to smarter wireless solutions

Content

Quick reference

Parameters

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. Some 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.

Additionaly, some functions can also use PIC STATUS flags. Refer to the PIC datasheet [8] (`STATUS` register) for details.

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	–TO = 0
Preconditions	–
Remarks	This is forced watchdog overflow reset. To distinguish the power down reset from this one the `-TO` (`STATUS.4`) status flag is intended. `-TO` = 0 after `reset()` execution `-TO` = 1 after power up Refer to the PIC datasheet [8] (`STATUS` register) for details.
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();`

iqrfSleep

Function	Entering the 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	The STATUS flags `-PD` (`STATUS.3`) and `-TO` (`STATUS.4`) are setup by this function: `-TO` = 1, `-PD` = 0
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 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	There are no restriction concerning Sleep and wake-up due to OS from OS v2.03. All features are under user's control. See example E01-TR [10].
Side effects	Global interrupt enable (GIE) is controlled by OS again after wake-up.
See also	–

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`
	`// It was default disabled by OS up to v2.03`
	`// automatically enabled before iqrfSleep`
	`// and automatically disabled after iqrfSleep`
	`// Now wake-up on pin change is fully under user control`
	`iqrfSleep();  // Put the module into Sleep mode`
	`RBIF = 0;  // Clear flag`
	`if (buttonPressed) // If button is pressed`
	`{ ... }  //  ...`

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-02/03) 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 in respective TRIS bits (C7 in, the others out). 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 displayd here according the result`

batteryValueOK

Function	Get a reference value for battery check from EEPROM.
Purpose	Power supply measurement to check whether the voltage is sufficient, especially for battery applications.
Syntax	`uns8 batteryValueOK()`
Parameters	–
Return value	–
Output values	Return value is factory calibrated constant with respect to piece-to-piece variability of device parameters.
Preconditions	• The `THRESHOLD` voltage must be selected according to desired limit. • Battery check is based on A/D measurement. See example E10-BATTERY [10].
Remarks	Power supply (on the C3 SIM pin) is measured by internal A/D converter of the microcontroller. Supply voltage depending on the battery condition is used as a reference while the measured input voltage is stable (LED voltage drop). The A/D converter must be controlled via the user program. The A/D result is stored in the `ADRESH` special function PIC register.
Side effects	–
See also	–

Example	`// #define THRESHOLD    1        // 3.0 V`
	`// #define THRESHOLD    5        // 2.9 V`
	`   #define THRESHOLD   10        // 2.8 V  for BL-ER14250 battery`
	
	`      ...                        // include A/D power measurement here`
	`                                 // A/D result is in ADRESH`
	
	`pomi = batteryValueOK();          // Get calibrated value from EEPROM`
	`if (pomi > ADRESH)                 // Comparison`
	`  if (pomi-ADRESH > THRESHOLD)    // Exhausted?`
	`    _LED = 1;                     //   Yes: LED on`

getTemperature

Function	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]. Temperature accuracy can be improved from ±2 °C (typical) to ±0.1 °C by individual calibration. Refer to IQRF support [6].
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`

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`. 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`

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.`

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`.
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, RFRXpacket or RFTXpacket` functions are active.
See also	`isDelay, 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` should be used before.
Remarks	• The 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`
Example	

```
     // 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
```

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` **`setOffPulsingLEDR(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`
	`setOnPulsingLEDR(25);  // 250 ms On`
	`setOffPulsingLEDR(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 (TRISB.7).
Side effects	• The appropriate PIC pin is not restored to the state before pulsingLEDR/pulseLEDR (TRISB.7 == 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 (TRISB.7 == 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 (TRISB.6).
Side effects	• The appropriate PIC pin is not restored to the state before pulsingLEDG/pulseLEDG (TRISB.6 == 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 (TRISB.6 == 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 35)
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 (0 to 0xBF for Node, 0xA0 to 0xBF for Coordinator). 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].
Side effects	• Any attempt to write to protected area above 0xBF leads to writing to EEPROM location 0xA0.
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);  // Writing to protected area is redirected to 160 (0xA0)`
	`X = eeReadByte(160);      // X == 0x75 will be read after illegal writing`

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].
	  • (`0` to `0xBF - length + 1`) for Node
	  • (`0xA0` to `0xBF - length + 1`) for Coordinator
	• `uns8 length`: number of bytes to be written from `bufferINFO` (1 to 35)

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 writing to EEPROM location 0xA0.
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);  // EEPROM address 160 is used instead of protected area`
	`//  EEPROM[160] = bufferINFO[0]`
	`//  ...`
	`//  EEPROM[160] = bufferINFO[15]`

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);`

setPIR1

Function	Setup microcontroller flags in PIR1
Purpose	Access to control flags for PIC internal peripherals (I^2C, A/D, ...).
Syntax	`void` **`setPIR1(value)`**
Parameters	`uns8 value`: value to be written
Return value	–
Output values	–
Preconditions	–
Remarks	PIR1 belongs to protected registers not user accessible in standard ways for security reasons. Thus, to allow access to some PIC internal peripherals, the specialized function is available to setup the appropriate flags in the PIR1 control register. See PIC datasheets [8].
Side effects	–
See also	–
Example	`setPIR1(0)  // ADIF = 0; SSPIF = 0;`

Buffers, data blocks

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`
Example	`clearBufferINFO();`

copyBufferINFO2COM

Function	Copy `bufferINFO` to `bufferCOM`
Purpose	Data transfer between buffers
Syntax	`void` **`copyBufferINFO2COM()`**
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	35 B is copied. See example E06 - RAM [10].
Side effects	–
See also	`clearBufferINFO, copyBufferINFO2RF, copyBufferRF2COM, copyBufferRF2INFO, copyBufferCOM2RF, copyBufferCOM2INFO, compareBufferINFO2RF, copyMemoryBlock`
Example	`copyBufferINFO2COM();`

copyBufferINFO2RF

Function	Copy `bufferINFO` to `bufferRF`
Purpose	Data transfer between buffers
Syntax	`void` **`copyBufferINFO2RF()`**
Parameters	–
Return value	–
Output values	–
Preconditions	–
Remarks	35 B is copied. See example E06 - RAM [10].
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	–
Remarks	35 B is copied. See example E06 - RAM [10].
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	–
Remarks	35 B is copied. See example E06 - RAM [10].
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	–
Remarks	35 B is copied. See example E06 - RAM [10].
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	–
Remarks	35 B is copied. See example E06 - RAM [10].
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].
Side effects	–
See also	`clearBufferINFO, copyBufferINFO2RF, copyBufferRF2INFO`
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. 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]`

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 "`
	`appInfo();` `// Dynamic change of application data`
	`bufferINFO[29] = '2';`
	`eeWriteData(__EEAPPINFO+29,1); // #01 changed to #02`

SPI

enableSPI

Function	Activate SPI communication module and related pins
Purpose	Enable SPI communication
Syntax	`void` **`enableSPI()`**
Parameters	–
Return value	SPI Status is switched to *SPI ready* (communication mode).
Output values	–
Preconditions	–
Remarks	• The PIC internal SPI hardware module and appropriate pins (C5 – C8) are cofigured and activated as SPI Slave. • Take into account that Master is allowed to send data whenever SPI is active after `enableSPI`. • See SPI Implementation in the IQRF platform [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`
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	SPI Status is switched to *SPI not active*.
Output values	–
Preconditions	–
Remarks	The PIC internal SPI hardware module is disabled and related pins (C5 – C8) are reconfigured to general I/Os. See SPI Implementation in the IQRF platform [5] and example E07-SPI [10].
Side effects	• The appropriate PIC pins need not be 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`
Example	`See getStatusSPI`

startSPI

Function	Indicate data ready to Master and provide data from `bufferCOM` to Master according to Master's clock (typically on OS background).
Purpose	Initiate SPI packet transmission from Slave (request to Master)
Syntax	`void` **`startSPI(length)`**
Parameters	`uns8 length`: number of bytes to be sent (0 to 35)
Return value	–
Output values	SPI Status is switched to *SPI data ready*.
Preconditions	SPI must be enabled by the `enableSPI` function before.
Remarks	• SPI runs on OS background except of the Slow mode case (during RF receiving). • `startSPI(0)` is useful for recovering SPI from communication failures (e.g. the CRC mismatch). • See SPI Implementation in the IQRF platform [5] and example E07-SPI [10].
Side effects	–
See also	`enableSPI, disableSPI, stopSPI, getStatusSPI`

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	`See getStatusSPI`

stopSPI

Function	Stop SPI communication
Purpose	Suspend SPI transmissions whenever it suits to Slave (e.g. not to lose previous data in `bufferCOM`)
Syntax	`void` **`stopSPI()`**
Parameters	–
Return value	–
Output values	SPI Status is switched to *User stop*.
Preconditions	–
Remarks	• SPI transmission is stopped but SPI remains active (enabled). Communication can continue after next `startSPI`. • `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 platform [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`
Example	`See getStatusSPI`

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	`param1`: 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	See SPI Implementation in the IQRF platform [5] and example E07-SPI [10].
Side effects	–
See also	`enableSPI, disableSPI, startSPI, stopSPI`
Example	

```
     // 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:
    PacketLength = param1;  //    Store packet length to user variable
    stopSPI();              //    Prohibit Master from transmitting
    copyBufferCOM2INFO();   //    until received packet is stored by the user
    startSPI(0);            //    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.
```

RF

setTXpower

Function	Set RF output power
Purpose	Change RF range
Syntax	`void` **`setTXpower(level)`**
Parameters	`uns8 level:` 1 (min.) to 7 (max.)
Return value	–
Output values	–
Preconditions	–
Remarks	Actual output power is non-linear and depends on TR type. See example E03–TR [10].
Side effects	–
See also	`RFTXpacket`
Example	`setTXpower(7);  // Max. RF output power`

RFTXpacket

Function	Send RF packet of specified length from `bufferRF` to specified addresse
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 and packet length. • To enable comfortable acknowledge, routing vector used by Coordinator is delivered to Node in reversed order ready for immediate usage. To take advantage of this the communication should be initiated by Coordinator. • See examples E01–TX, E03–TR and E09–LINK [10].
Side effects	• OPTION.7 = 1 (Port B pull-ups disabled) • bufferRF[DLEN] and bufferRF[DLEN+1] are destroyed
See also	`RFRXpacket, setTXpower` 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)
setNetworkingOff();
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)
setNetworkOne();      // Select Network 1. The NTWF flag (PIN.7) is set here.
// setCoordinatorMode(); Select Coordinator mode - not necessary
                      // in this OS versions (already done by setNetworkOne)
bufferRF[0] = "I";    // Data to send
bufferRF[1] = "Q";
DLEN = 2;             // 2 B packet
RX = 10;              // Packet for Node #10
// _ROUTF = 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 via Nodes 1,2,3 and 4
PIN = 0;              // PIN preclearing (update also after every RFRXpacket
                      // before every RFTXpacket)
setNetworkOne();      // Select Network 1. The NTWF flag (PIN.7) is set here.
// setCoordinatorMode(); Select Coordinator mode - not necessary
                      // in this OS versions (already done by setNetworkOne)
bufferRF[0] = "I";    // Data to send
bufferRF[1] = "Q";
DLEN = 2;             // 2 B packet
RX = 10;              // Packet for Node #10
_ROUTF = 1;           // Routing enabled for outgoing packets
// RTDEF = 0;            Standard routing, 4+1 hops. Reserved
                      // for future OS versions (various routing types)
RTV[0] = 1;           // Routing vector. This will not be necessary to
RTV[1] = 2;           // specify in future OS versions.
RTV[2] = 3;
RTV[3] = 4;
RFTXpacket();         // Send the packet to IQMESH Node #10 in this network
                      // Reception depends on the Node (its current network
                      // or filtering)
                      // Routing vector is reversed (4,3,2,1) by OS for Node
```

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	`DLEN =` packet length `_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 in number of 10 ms ticks should be specified in `toutRF` (1 to 255) • 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 does not terminate the reception. • 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 `setFilteringOn` and `setFilteringOff`. • See examples E02–RX, E03–TR, E09–LINK and E11-MEASUREMENT [10].
Side effects	• Update PIN before every `RFTXpacket` folowed after `RFRXpacket`. • SPI is switched to Slow mode during `RFRXpacket` activity. • Result of `captureTicks` is destroyd if `startCapture` is active on background at the same time. • OPTION.7 = 1 (Port B pull-ups disabled). • OPTION.6 = 1 (Interrupt on rising edge of INT pin). See PIC datasheet (Option register) [8]. • bufferRF[DLEN] and bufferRF[DLEN+1] is destroyed.
See also	`RFTXpacket` and (in case of IQMESH) also other RF functions

Example1	`// Peer-to-peer topology`
	`toutRF = 10; // RF timeout 100 ms`
	`if RFTXpacket(); // Try to receive RF packet. Packet received?`
	`// Program stays here until the packet is received`
	`// or the timeout is expired.`
	`{ // Yes:`
	`copyBufferRF2INFO; // Store received data`
	`PacketLength = DLEN; // and possibly other info (packet length, ...)`
	`}`
	`else`
	`{ // No:`
	`... // Timeout expired. Arrange respective operations.`
	`}`

Example2	IQMESH: See `setNodeMode` and `setNetworkFilteringOn`.

Networking

setNetworkOne

Function	Select IQMESH networking, Network 1
Purpose	Switch (from IQMESH Network 2 or Peer-to-peer) to IQMESH Network 1
Syntax	`void` **`setNetworkOne()`**
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH only. Default OS mode is Peer-to-peer.
Remarks	• IQMESH allows every TR module to work in more independent networks. This OS version supports 2 networks and working as a Coordinator in Network 1 and as a Node in Network 2. • See example E11 - DATACENTER [10]. • This settings affects both `RFRXpacket` and `RFTXpacket` features.
Side effects	TR module is switched to Coordinator mode by this function in this OS version. Thus, `setNetworkOne` is equivalent to `setCoordinatorMode` in this OS version. This is not guaranteed for future OS versions.
See also	`setNetworkTwo, setNetworkingOff, setCoordinatorMode, setNodeMode`
Example	See `setNodeMode`

setNetworkTwo

Function	Select IQMESH networking, Network 2
Purpose	Switch (from IQMESH Network 1 or Peer-to-peer) to IQMESH Network 2
Syntax	`void` **`setNetworkTwo()`**
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH only. Default OS mode is Peer-to-peer.
Remarks	• IQMESH allows every TR module to work in more independent networks. This OS version supports 2 networks and working as a Coordinator in Network 1 and a Node in Network 2. • See example E11 - MEASUREMENT [10]. • This settings affects `RFRXpacket` and `RFTXpacket` features.
Side effects	TR module is switched to Node mode by this function in this OS version. Thus, `setNetworkTwo` is equivalent to `setNodeMode` in this OS version. This is not guaranteed for future OS versions.
See also	`setNetworkOne, setNetworkingOff, setCoordinatorMode, setNodeMode`
Example	See `setNodeMode`

setNetworkingOff

Function	Select Peer-to-peer mode
Purpose	Switch from IQMESH to Peer-to-peer
Syntax	`void` **`setNetworkingOff()`**
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	`setNetworkOne, setNetworkTwo, setCoordinatorMode, setNodeMode`

Example	`setNetworkOne();  // IQMESH selected`
	`  ...  // TR is assigned as a Coordinator and communicates`
	`  // in IQMESH networking mode here`
	`  //`
	`setNetworkingOff();  // Switch to Peer-to-peer mode`
	`  ...  // Now TR communicates without networking support`

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	TR module is switched to IQMESH, Network 1 in this OS version. Thus, `setCoordinatorMode` is equivalent to `setNetworkOne` in this OS version. This is not guaranteed for future OS versions.
See also	`setNetworkOne, setNetworkTwo, setNodeMode`
Example	See `setNodeMode`

setNodeMode

Function	Set Node mode
Purpose	Assign the TR module as a 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	TR module is switched to IQMESH, Network 2 in this OS version. Thus, `setNodeMode` is equivalent to `setNetworkTwo` in this OS version. This is not guaranteed for future OS versions.
See also	`setNetworkOne, setNetworkTwo, setNodeMode`
Example	

```
   // Forwarding a packet from Network 2 to Network 1
setNetworkTwo();       // These 2 functions are equivalent in this OS version
// setNodeMode();         In either case the result is the same.
if (RFRXpacket())      // Receive a packet in Network 2
{
  setNetworkOne();      // Then switch to Network 1
// setCoordinatorMode();  These 2 functions are equivalent in this OS version
  RX=5;
  RFTXpacket();         // and forward the packet to addressee 5 in Network 1
}
```

setNetworkFilteringOn

Function	Start filtering incoming non-networking packets and packets coming from non-current network(s).
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 networks).
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`

setLoggingOn

Function	Start logging communication with adjacent devices
Purpose	To keep track of devices in range
Syntax	`void` **`setLoggingOn()`**
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH only. Default OS condition is Logging off. **Not implemented**. Reserved for future OS versions (Discovery service etc.). **Do not use in this OS version.**
Remarks	–
Side effects	–
See also	`setLoggingOff`

Example	`setLoggingOn();  // Start logging`
	`  ...  // All communication with adjacent devices is logged now`

setLoggingOff

Function	Stop logging communication with adjacent devices
Purpose	Not to track of devices in range
Syntax	`void` **`setLoggingOff()`**
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH only. Default OS condition is Logging off. **Not implemented**. Reserved for future OS versions (Discovery service etc.). **Do not use in this OS version.**
Remarks	–
Side effects	–
See also	`setLoggingOn`

Example	`setLoggingOff();  // Stop logging`
	`  ...  // No communication with adjacent devices is logged now`

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 - MEASUREMENT [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	Outgoing packets will be delivered via routing devices on background.
Syntax	`void` **`setRoutingOn()`**
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH only. Default OS condition is Routing off. **Not implemented**, reserved for future OS versions. **Do not use in this OS version.**
Remarks	To select wheter the transmitting packet should be delivered via routing devices use the `_ROUTEF` flag (`PIN.5`) instead of this function: • `_ROUTEF = 1`: routing enabled. Routing vector RTV[0 to 3] should be specified. • `_ROUTEF = 0`: routing disabled.
Side effects	–
See also	`setRoutingOff`
Example	See `RFTXpacket`, example 3

setRoutingOff

Function	Routing disabled
Purpose	Outgoing packets will not be delivered via routing devices on background.
Syntax	`void` **`setRoutingOff()`**
Parameters	–
Return value	–
Output values	–
Preconditions	For IQMESH only. Default OS condition is Routing off. **Not implemented**, reserved for future OS versions. **Do not use in this OS version.**
Remarks	To select wheter the transmitting packet should be delivered via routing devices use the `_ROUTEF` flag (`PIN.5`) instead of this function: • `_ROUTEF = 1`: routing enabled. Routing vector RTV[0 to 3] should be specified. • `_ROUTEF = 0`: routing disabled.
Side effects	–
See also	`setRoutingOn`
Example	See `RFTXpacket`, example 2

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) 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 - MEASUREMENT, E11 - DATACENTER [10] and IQMESH specification [4]. This function is active until successfully finished or fixed 10 s timeout expired. RF power is not affected (from OS v2.10).
Side effects	The following values are modified and not restored: • WDTCON • DLEN • PIN • toutRF • OPTION.7 = 1 (Port B pull-ups disabled) • OPTION.6 = 1 (Interrupt on rising edge of PIC INT pin) • bufferRF[0 to 63] = 0 • bufferINFO[0 to 35] = 0 • Result of `captureTicks` is destroyd if `startCapture` is active on background at the same time.
See also	`bondNewNode, amIBonded, removeBond, rebondNode, wipeBondNR, getNetworkParams`

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`
	`}`
	`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` or `wipeBondNR`)
	• 0 – Node is not bonded:
	  • no `bondRequest` has ever been successfully executed
	  • after `removeBond` or `wipeBondNR`

Output values	–
Preconditions	For IQMESH only. Result is not depended on the Coordinator at all.
Remarks	See example E11 - MEASUREMENT [10].
Side effects	–
See also	`bondRequest, removeBond, wipeBondNR`

Example	`while (!amIBonded())  // Request for beeing bonded (if not bonded yet)`
	`{`
	`  bondRequest();  // Repeatedly try to bond`
	`  clrwdt();`
	`}  // until successful`
	`setTXpower(7);  // Max. RF output power for future communication`

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. Max. 239 Nodes can be bonded to single network.
Remarks	• See example E11 - MEASUREMENT [10]. • For rebonding use bondRequest again (reserved Node number will be offered to Coordinator then). The Coordinator accepts it if this number is still free. • 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, wipeBondNR`

Example	`removeBond(); // Remove the bond. Node number remains reserved for this`
	`// Node to be bonded with the same Node number in the future.`

wipeBondNR

Function	Unbond the Node from the network and cancel the Node number reservation for given Node. Record it to EEPROM.
Purpose	Exclude the Node from the network on Node's side and wipe the Node number (not to keep it for the future)
Syntax	`void` **`wipeBondNR()`**
Parameters	–
Return value	–
Output values	• The `amIBonded` function `starts` to return value == 0 whenever is called until the Node is bonded again. • Additionally, the Node number is wiped not supporting future rebonding with the same Node number. • Coordinator is not affected at all.
Preconditions	For IQMESH only
Remarks	The number is released for future bonding to any Node. See example E11 - MEASUREMENT [10]. The effect will not surfice in Coordinator until `bondRequest/bondNewNode` procedure is invoked.
Side effects	
See also	`removeBond, rebondNode, bondNewNode`

Example	`wipeBondNR(); // Remove the bond and release the Node number to be used`
	`// by any Node in the future.`

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()`**
Parameters	–
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 • The `isBondedNode` function starts to return value == 1 whenever is called while the Node is in the list of bonded Nodes.
Preconditions	• For IQMESH 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 - DATACENTER [10]. • 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. • In case of rebonding the original (reserved) Node number is prefered to be assigned. If it is just occupied by another Node a different Node number is assigned. • RF power is not affected (from OS v2.10).
Side effects	The following values are modified and not restored: • WDTCON modified • toutRF modified • OPTION.7 = 1 (Port B pull-ups disabled) • OPTION.6 = 1 (Interrupt on rising edge of INT pin) • bufferRF[0 to 63] = 0 • bufferINFO[0 to 35] = 0 • Result of `captureTicks` is destroyd if `startCapture` is active on background at the same time.
See also	`bondRequest, removeBondedNode, rebondNode, isBondedNode`

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

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 - DATACENTER [10]. Coordinator will start to assign Node numbers from 0 for Nodes with wiped Node numbers. For not wiped ones will try to restore the original Node numbers.
Side effects	• bufferINFO[0 to 35] = 0
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=83
2 **RAM map** and **EEPROM map**, IQRF OS User's guide, Appendix [1]
3 **IQRF** home page 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-31B** datasheet: www.iqrf.org/weben/downloads.php?id=92
TR-32B datasheet: www.iqrf.org/weben/downloads.php?id=94
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

- 100118 OS v2.10, for TR-31B and TR-32B
  Bugs in `readFromRAM` examples 4 and 5 corrected
  Documentation and Information updated
- 091201 User's/Reference guide supplement for OS v2.10
- 090819 OS v2.09

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