IQRF OS

Operating System

Version 3.07D
for (DC)TR-7xD

Reference Guide

Smarter wireless. Simply.

Quick reference

This document is valid for TR as well as DCTR transceivers. For simplicity, only TR is used further on throughout the document (with minor exceptions where needed).

Values between system functions and superordinate program are passed on via parameters. OS uses 3 parameters in total: `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.

Five stack levels are available to call all OS functions in subroutines.

Unless otherwise stated, OS functions run in OS foreground. Thus, the program continues not until the function is finished.

Several functions, e.g. `startSPI(x)` or `startDelay(x)` run in OS background. Thus, they are not blocking. The program execution continues immediately further and the user can check the result later on.

Functions

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

OS functions

Control

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 (for TR modules in SIM format, e.g. TR-72D) or Q12 (for TR-76D) 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, see the Example 2 below. Wake-up on pin change is default disabled. • This function is not time-efficient for subsequent short sleep periods, especially if RF IC is off. For faster operation in such cases use `sleep()` instead but you should ensure minimal consumption by user program. See Example 3.
Remarks	• IOCBF flag is cleared automatically by OS. • Flags `IOCBN` and `IOCBP` are unchanged (not cleared) within `iqrfSleep`. • Wake-up types can be identified via the `-TO` and `-PD` status flags (in the MCU `Status` register).
Side effects	Global interrupt enable (GIE) is controlled by OS again after wake-up.
See also	`setRFsleep`

Example 1

```
      // 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
```

Example 2

```
  // Wake-up on pin change. See example E01-TX and IQRF-macros.h header file.
GIE = 0;                              // Disable all interrupts
writeToRAM(&IOCBN, IOCBN | 0x10);     // Negative edge active.
                                      // Instead of IOCBN.4=1;
                                      // Bit IOCBN.4 cannot be written
                                      // directly due to OS restriction.
IOCBP.4 = 1;                          // Positive edge active
IOCIE = 1;                            // Interrupt on change enabled
SWDTEN = 0;                           // Watchdog disabled
iqrfSleep();                          // Sleep
GIE = 0;
writeToRAM(&IOCBN, IOCBN & 0xEF);   // Clear negative edge flag (IOCBN.4)
IOCBP.4 = 0;                          // Clear positive edge flag
GIE = 1;
if (buttonPressed)                    // If button is pressed
  { ... }                             //   ...
```

Example 3

```
iqrfSleep();        // Sleep
  ...
  ...               // Wake-up, RF IC remains off
stopLEDR();         // Disable peripherals to minimize consumption
sleep();            // Faster (if RF IC is off). This is not an IQRF function
                    // but a machine instruction supported by C compiler.
pulseLEDR();        // Continue after wake-up
```

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	• RF IC is set off. • OS system clock (ticks) are derived from MCU internal RC oscillator instead of precise RF IC crystal.
Preconditions	–
Remarks	• Wake-up can be caused by `setRFready`, `RFTXpacket`, `RFRXpacket or checkRF` • Refer to the datasheet of given TR module for power consumption saving.
Side effects	–
See also	`setRFready`, `iqrfSleep`, `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 and precise ticks.
Syntax	`void` **`setRFready()`**
Parameters	–
Return value	–
Output values	• RF IC is set on (the RF ready mode) but RX chain still stays off (unlike the RX mode). See IQRF User's guide, RF IC modes. • RF IC crystal oscillator starts up. • OS system clock (tick) is based on precise RF IC crystal oscillator instead of MCU internal RC one.
Preconditions	–
Remarks	After the RF wake-up the RX chain can be set on faster which enables faster `checkRF(x)`, `RFRXpacket() or RFTXpacket()`.
Side effects	–
See also	`setRFsleep, iqrfSleep, checkRF, RFTXpacket, RFRXpacket`

Example	`setRFready();  // Wake the RF circuitry up from RF sleep in advance`
	`...`
	`RFTXpacket();  // for immediate response`

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-04x) 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`
	  • Timer6 is not automatically stopped and user interrupt is not automatically disabled in debug.
	  • When entering debug, the application must not have enabled interrupt from any of user peripherals.
	  • Debug must not be used within the user interrupt routine.

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 Breakpoint* 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`, `memoryOffsetTo`, `memoryOffsetFrom` and `memoryLimit` are not displayed • Watchdog is cleared while in Debug mode
See also	–

Example 1

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

Example 2

```
// Similar as Example 1 but utilizing macro breakpoint.
// See header file IQRF-macros.h.
if (compareBufferINFO2RF(4))
  {
     breakpoint(1);     // match
  }
else
  {
     breakpoint(2);     // mismatch
  }
        // Skip Breakpoint 1 or 2 will be displayed here according the result
```

getSupplyVoltage

Function	Power supply measurement (up to 3.84 V)
Purpose	Battery check
Syntax	`uns8 getSupplyVoltage()`
Parameters	–
Return value	level = 1, 2, …59 `Voltage [V] = 261.12 / (127 - level)`
Output values	–
Preconditions	–
Remarks	• Internal power supply voltage is checked. • In case of TR modules with LDO 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.
Side effects	A/D converter control registers are changed.
See also	–

Example	`if (getSupplyVoltage() < 38)`
	`  …  // Low battery`
	`else`
	`  …  // Voltage > 2.93 V`

getTemperature

Function	Read temperature from on-board sensor
Purpose	Temperature measurement
Syntax	`uns8` **`getTemperature()`**
Parameters	–
Return value	• Temperature in °C, integer part, not rounded • Negative temperatures are in two's complement format (e.g. `0xFB` means -5 °C) • 0x80 (-128 °C) indicates an error in communication with temperature sensor (temperature sensor damaged or not present, i.e. for TR modules without the “T” postfix, e.g. TR-72D.
Output values	`param3`: 12 b output value of the sensor in 0.0625 °C units. Thus, upper 8 b represent the integer part of the temperature and lower 4 b represent the fractional part. The resolution is limited to 0.5 °C, therefore the lowest 3 b are always cleared. Negative temperatures are in the two's complement format. See datasheet of the temperature sensor. Examples:

Temperature	Return value	param3
50 °C	0x32	0x320
5 °C	0x05	0x050
5.5 °C	0x05	0x058
0.5 °C	0x00	0x008

Temperature	Return value	param3
0 °C	0x00	0x000
-0.5 °C	0xFF	0xFF8
-1 °C	0xFF	0xFF0
-8 .5 °C	0xF7	0xF78

Preconditions	• Applicable for TR modules with MCP9808 (Microchip) temperature sensor only (e.g. not for TR-76D). • 300 ms delay is required in LP or XLP RX mode or after wake up from sleep
Remarks	• Resolution 0.5 °C, accuracy: 0.5 °C • Takes about 3 ms. • See example E08–TEMPERATURE [10]
Side effects	–
See also	–

Example 1

```
   // For positive temperatures only
uns8   tempInt;                     // Temperature, integer part
uns8   tempFract;                   // Temperature, fractional part
tempInt = getTemperature();
tempFract = param3.low8 & 0x0F      // Temperature == tempInt + tempFract/16
                                    // Temperature == param3 * 0.0625 in °C
```

Example 2

```
   // Either positive or negative temperatures, fractional part ignored
T = getTemperature();          // Integer part of temperature

if (T >= 0x80) {
   sign = "-";                 // Negative
   T = (T ^ 0xFF) + 1;         // Get absolute value in °C
}
else
   sign = "+";                 // Positive
```

Example 3

```
   // Either positive or negative temperatures, with fractional part
if (getTemperature() >= 0x80) {
    sign = "-";                // Negative
    T = (param3 ^ 0xFFF) + 1; // Get absolute value, in 0.0625°C units
  }
else
   sign = "+";                 // Positive
```

Example 4

```
   // Temperature measurement after wake-up from sleep
iqrfSleep();
waitDelay(30);                      // 300 ms delay required
T = getTemperature();
```

Active (blocking) 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 periods (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 must 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].
See also	`waitMS`, `startDelay, startLongDelay`

Example 1	`  // 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();  // To generate a 10 ms pulse as precise as possible`
	`IO1 = 1;`
	`  ...  // Something shorter than 10 ms`
	`waitNewTick();  // 10 ms`
	`IO1 = 0;`

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`, `RFRXpacket` and `RFTXpacket`.
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`, `RFRXpacket` and `RFTXpacket`. • Internal ticks are based on internal RC oscillator if RF IC is sleeping. Thus, the delay can vary with temperature etc. in this case. See respective PIC datasheet [8].
See also	`startCapture`, `setRFready`

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. Non-blocking alternative to `waitDelay`.
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. Expiration can be checked by the `isDelay` function.
Side effects	• This function does not work properly if the `waitDelay` function is active. • Functionality is affected by `bondRequest`, `bondNewNode`, `RFRXpacket` and `RFTXpacket`. • Internal ticks are based on internal RC oscillator if RF IC is sleeping. Thus, the delay can vary with temperature etc. in this case. See respective PIC datasheet [8].
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`.
Remarks	The Delay timer measures specified time period on OS background. Expiration can be checked by the `isDelay` function.
Side effects	• This function does not work properly if the `waitDelay` function is active. • Functionality is affected by `bondRequest`, `bondNewNode`, `RFRXpacket` and `RFTXpacket`. • Delay in first tick can vary from 0 ms to 10 ms. If complete 10 ms is needed also in the first tick, use `waitNewTick` firstly. • Internal ticks are based on internal RC oscillator if RF IC is sleeping. Thus, the delay can vary with temperature etc. in this case. See respective PIC datasheet [8].
See also	`isDelay, startDelay, waitDelay`
Example	See `isDelay`

isDelay

Function	Information whether the Delay timer has expired
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`

Example 1	`// 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`

Example 2	`// 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. • Do not combine this function with direct access to LED pin (see `pulseLEDR` *Remarks*). If omitted, the pin state can be modified in background.
Remarks	• Blinking continues until it is stopped by the user (e.g. by `stopLEDR`).
Side effects	–
See also	`setOnPulsingLED, setOffPulsingLED, stopLEDR, pulseLEDR`
Example 1	`pulsingLEDR();  // continuous blinking on OS background`

Example 2	`// 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. • Do not combine this function with direct access to LED pin (see *Remarks*). If omitted, the pin state can be modified in background.
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), e.g. `_LEDR = 1`.
Side effects	–
See also	`setOnPulsingLEDR, pulsingLEDR, stopLEDR`

Example	`setOnPulsingLED(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	Do not combine this function with direct access to LED pin (see `pulseLEDR` *Remarks*). If omitted, the pin state can be modified in background.
Remarks	–
Side effects	–
See also	`pulsingLEDR, pulseLEDR`

Example 1	`pulsingLEDR();  // Start blinking on OS background`
	`  ...  // Blinking continues during any operation`
	`stopLEDR();  // Stop blinking`

Example 2	`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`

Example 3	`_LEDR = 1;  // LEDR on`
	`  ...`
	`stopLEDR();  // LEDR off`

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. • Do not combine this function with direct access to LED pin (see `pulseLEDG` *Remarks*). If omitted, the pin state can be modified in background.
Remarks	Blinking continues until it is stopped by the user (e.g. by `stopLEDG`).
Side effects	–
See also	`setOnPulsingLED, setOffPulsingLED, stopLEDG, pulseLEDG`
Example 1	`pulsingLEDG();  // continuous blinking on OS background`

Example 2	`  // 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. • Do not combine this function with direct access to LED pin (see *Remarks*). If omitted, the pin state can be modified in background.
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), e.g. `_LEDG = 1`.
Side effects	–
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	Do not combine this function with direct access to LED pin (see `pulseLEDG` *Remarks*). If omitted, the pin state can be modified in background.
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). • Possible user LEDR pin level (in PORT or LATCH register) changed in foreground can be overriden in background.
See also	`pulsingLEDG, pulseLEDG`

Example 1	`pulsingLEDG();  // Start blinking on OS background`
	`  ...  // Blinking continues during any operation`
	`stopLEDG();  // Stop blinking`

Example 2	`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`

MCU EEPROM

eeReadByte

Function	Read one byte from specified location in EEPROM
Purpose	Access to EEPROM
Syntax	`uns8` **`eeReadByte(address)`**
Parameters	`uns8 address`: address in EEPROM (0 to 0xBF). See EEPROM map [2].
Return value	• Value (0 to 255) read from specified EEPROM location • 0 when attempted to read from address 0xC0 or higher
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	–
See also	`eeReadData, eeWriteByte, eeWriteData`
Example 1	`i = eeReadByte(0);  // Copy 1 byte from EEPROM from address 0 to i`
Example 2	`// Illegal access: Avoid access to EEPROM locations 0xC0 or higher` `i = eeReadByte(0xC8);  // Reading from protected area is redirected to 0xA0`

eeReadData

Function	Read a block of specified length from specified location in EEPROM to `bufferINFO`
Purpose	Block access to EEPROM
Syntax	`void` **`eeReadData(address, length)`**
Parameters	• `uns8 address`: 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]` • `bufferINFO[0 to length - 1]` is cleared when attempted to read from address 0xC0 or higher
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	–
See also	`eeReadByte, eeWriteByte, eeWriteData`

Example 1	`eeReadData(0x0A, 16); // copy 16B from EEPROM from address 0x0A to bufferINFO`
	`// bufferINFO[0] = EEPROM[0x0A]`
	`// ...`
	`// bufferINFO[15] = EEPROM[0x19]`

Example 2	`// Illegal access: Avoid access to EEPROM locations 0xC0 or higher`
	`eeReadData(0xC8, 16); // EEPROM address 0xA0 used instead of protected area`
	`// bufferINFO[0] = EEPROM[0xA0]`
	`// ...`
	`// bufferINFO[15] = EEPROM[0xA0]`

eeWriteByte

Function	Write one byte to specified location in EEPROM
Purpose	Access to EEPROM
Syntax	`void` **`eeWriteByte(address, data)`**
Parameters	• `uns8 address`: 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`
Example 1	`eeWriteByte(0xBF, 0x75)  // store 0x75 to EEPROM to address 0xBF` `eeWriteByte(0x80, myVar)  // copy myVar to EEPROM to address 0x80`
Example 2	`// Illegal access: Avoid access to EEPROM locations 0xC0 or higher` `eeWriteByte(0xC6, 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(address, length)`**

Parameters	• `uns8 address`: 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`

Example 1	`eeWriteData(0x0A,16); // copy 16B from bufferINFO to EEPROM to address 0x0A`
	`//  EEPROM[0x0A] = bufferINFO[0]`
	`//  ...`
	`//  EEPROM[0x19] = bufferINFO[15]`

Example 2	`// Illegal access: Avoid access to EEPROM locations 0xC0 or higher`
	`eeWriteData(0xC8,16); // Attempt to write to protected area – nothing is`
	`// written.`

Serial EEPROM

eeeReadData

Function	Read a data block of specified length from specified location in serial EEPROM to `bufferINFO`
Purpose	Read from serial EEPROM
Syntax	`bit` **`eeeReadData(address)`**
Parameters	• `uns16 address`: initial address in serial EEPROM (`0` to `0x7FFF`).
Return value	• `1` Read successful • 0 Read unsuccessful (e.g. due to damaged or not populated memory device). Additionally, the _eeeError flag is set.
Output values	`bufferINFO[0` to 63`]`
Preconditions	• `memoryLimit` specifies number of bytes (1 to 64) to be read. It must be set before every `eeeReadData` call. If `memoryLimit` = 0, complete 64 B is read. • To respect accessible range, the following rule must be observed: `address + memoryLimit < 0x8000`. See Example 2 and 3. • Do not use for Coordinator in networks utilizing Discovery.
Remarks	• Memory range `0` to `0x7FFF` is accessible. • `memoryLimit` is automatically cleared after every `eeeReadData` call.
Side effects	–
See also	`eeeWriteData`

Example 1	`// To read 64 B`
	`// When memoryLimit is kept cleared from previous operations`
	`eeeReadData(0x3C); // copy 64B from serial EEPROM from address 0x3C to`
	`// bufferINFO`
	`// bufferINFO[0] = serial EEPROM[0x3C]`
	`// ...`
	`// bufferINFO[63] = serial EEPROM[0x7B]`

Example 2	`memoryLimit = 40; // To read 40 B`
	`eeeReadData(0x3C); // copy 40B from serial EEPROM from address 0x3C to`
	`// bufferINFO`
	`// bufferINFO[0] = serial EEPROM[0x3C]`
	`// ...`
	`// bufferINFO[39] = serial EEPROM[0x63]`
	`// memoryLimit is automatically cleared here`

Example 3	`memoryLimit = 40; // To read 40 B`
	`eeeReadData(0x7FEE); // Illegal usage, out of 0x7FFF boundary`

Example 4	`if (eeeReadData(0x0A))`
	`X = bufferINFO[0]`
	`else`
	`{`
	`... // Error handling`
	`}`

eeeWriteData

Function	Write a data block of specified length from `bufferINFO` to specified location in EEPROM
Purpose	Write to serial EEPROM
Syntax	`bit` **`eeeWriteData(address)`**

Parameters	• `uns16 address`: initial address in serial EEPROM
	  • For Coordinator in networks not utilizing Discovery and for Nodes: `0` to `0x7FFF`
	  • For Coordinator in networks utilizing Discovery: `0x800` to `0x7FFF`. This range may be a subject to change in future IQRF OS versions based on added functionality.

Return value	• `1` Write successful • 0 Write unsuccessful (e.g. due to damaged or not populated memory device). Additionally, the _eeeError flag is set.
Output values	Memory range `0` to `0x7FFF` is accessible.
Preconditions	• `memoryLimit` specifies number of bytes (1 to 64) to be written. It must be set before every `eeeWriteData` call. If `memoryLimit` = 0, complete 64 B is written. • `address` and `memoryLimit` must be selected so as the addressed space fits within a single 64 B long page, e.g. 0 – 0x3F, 0x40 – 0x7F, …, 0x7FFC0 – 0x7FFF. See Example 2, 3 and 4.
Remarks	`memoryLimit` is automatically cleared after every `eeeWriteData` call.
Side effects	–
See also	`eeeReadData`

Example 1	`// To write 64 B`
	`// When memoryLimit is kept cleared from previous operations`
	`eeeWriteData(0x40); // copy 64B from bufferINFO to serial EEPROM`
	`// from address 0x40`
	`// EEPROM[0x40] = bufferINFO[0]`
	`//  ...`
	`// EEPROM[0x7F] = bufferINFO[63]`

Example 2	`memoryLimit = 6;  // To write 6 B`
	`eeeWriteData(0x40); // copy 6B from bufferINFO to serial EEPROM from`
	`// address 0x40`
	`// EEPROM[0x40] = bufferINFO[0]`
	`//  ...`
	`// EEPROM[0x45] = bufferINFO[5]`
	`// memoryLimit is automatically cleared here`

Example 3	`// To write 64 B`
	`// When memoryLimit is kept cleared from previous operations`
	`eeeWriteData(0x41); // Illegal access, boundary 0x80 crossed`

Example 4	`memoryLimit = 6;  // to write 6 B`
	`eeeWriteData(0x2C7E); // Illegal access, boundary 0x2C80 crossed`

Example 5	`memoryLimit = 1; // To write 1 B`
	`bufferINFO[0] = 'A'`
	`if (!eeeWriteData(0x0A))`
	`  ...  // Error handling`

RAM

readFromRAM

Function	Read one byte from specified location in RAM
Purpose	Indirect access to RAM registers
Syntax	`uns8` **`readFromRAM(address)`**
Parameters	`uns16 address`: linear or traditional memory location address
Return value	Value read from specified location
Output values	–
Preconditions	–
Remarks	See example E06–RAM [10].
Side effects	`FSR0` register is modified.
See also	`writeToRAM, copyMemoryBlock, getINDF0, getINDF1`

Example	`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(address, value)`**
Parameters	• uns16 address: traditional or linear memory location address • `uns8 value`: value to be written
Return value	–
Output values	–
Preconditions	• Avoid writing to RAM areas dedicated to OS and to PIC special function registers otherwise OS can collapse. See RAM map [2].
Remarks	RAM can be accessed either directly (using common C commands like `X = Y;`) or indirectly. But indirect writng to the `INDFx` registers is not allowed. Due to security reasons all instructions writng to `INDFx` are removed during Upload. To avoid unintended behavior, all constructions writng to `INDFx` (either by the user or by the compiler) should be omitted. Instead of this IQRF OS provides complete support for indirect RAM addressing using extra system functions `readFromRAM`, `writeToRAM` and `copyMemoryBlock`. See example E06–RAM [10].
Side effects	`FSR0` register is modified.
See also	`readFromRAM, copyMemoryBlock, setINDF0, setINDF1`

Example 1	`  // Not allowed. The compiler uses INDFx in such cases.`
	`for (i=0; i<5; i++)`
	`  bufferRF[i] = i;`

Example 2	`  // Correct`
	`for (i=0; i<5; i++)`
	`  writeToRAM(bufferRF + i, i);`

setINDF0

Function	Write a value in the virtual `INDF0` register
Purpose	Indirect write to RAM
Syntax	`void` **`setINDF0(value)`**
Parameters	• `uns8 value`: value to be written
Return value	–
Output values	Register addressed by the `FSR0H` and `FSR0L` is modified
Preconditions	• `FSR0` (the `FSR0H` and `FSR0L` register pair) must be set before to define a destination. Traditional as well as linear address can be used. • Avoid writing to RAM areas dedicated to OS and to PIC special function registers otherwise OS can collapse. See RAM map [2].
Remarks	• Simple writing to the `INDF0` virtual register is not allowed. Due to security reasons all instructions using `INDF0` are removed during Upload. To avoid unintended behavior all constructions modifying `INDF0` (either by the user or by the compiler) should be omitted. Instead of this IQRF OS allows to write to indirectly addressed RAM using extra system function `setINDF0`. See example E06–RAM [10]. • Another possibility (but more code consuming) is using the `writeToRAM` function.
Side effects	–
See also	`setINDF1`, `getINDF0`, `getINDF1`, `writeToRAM`, `copyMemoryBlock`
Example	See `getINDF0 (by analogy with INDF1)`

setINDF1

Function	Write a value in the virtual `INDF1` register
Purpose	Indirect write to RAM
Syntax	`void` **`setINDF1(value)`**
Parameters	• `uns8 value`: value to be written
Return value	–
Output values	Register addressed by the `FSR1H` and `FSR1L` is modified
Preconditions	• `FSR1` (the `FSR1H` and `FSR1L` register pair) must be set before to define a destination. Traditional as well as linear address can be used. • Avoid writing to RAM areas dedicated to OS and to PIC special function registers otherwise OS can collapse. See RAM map [2].
Remarks	• Simple writing to the `INDF1` virtual register is not allowed. Due to security reasons all instructions using `INDF1` are removed during Upload. To avoid unintended behavior all constructions modifying `INDF1` (either by the user or by the compiler) should be omitted. Instead of this IQRF OS allows to write to indirectly addressed RAM using extra system function `setINDF1`. See example E06–RAM [10]. • Another possibility (but more code consuming) is using the `writeToRAM` function.
Side effects	–
See also	`setINDF0`, `getINDF0`, `getINDF1`, `writeToRAM`, `copyMemoryBlock`
Example	See `getINDF0`

getINDF0

Function	Read a value in the `INDF0` virtual register
Purpose	Indirect read from RAM
Syntax	`uns8` **`getINDF0()`**
Parameters	–
Return value	Value of the register addressed by the `FSR0H` and `FSR0L`
Output values	–
Preconditions	• `FSR0` (the `FSR0H` and `FSR0L` register pair) must be set before to define a destination. Traditional as well as linear address can be used.
Remarks	• See example E06–RAM [10]. • Additionally, simple reading the `INDF0` virtual register is allowed. This is even less memory consuming than `getINDF0`. See Example 2 below. • Another possibility is using the `readFromRAM` function. • `getINDF0` can advantageously be replaced by direct reading, e.g. `X = INDF0;` see Example 2.
Side effects	–
See also	`setINDF0`, `setINDF1`, `getINDF1`, `readFromRAM`, `copyMemoryBlock`

Example 1

```
      // Not allowed
FSR0 = ADDR_FROM;
FSR1 = ADDR_TO;
for(i = 0; i < 16; i++)
{
  X = INDF0;            // Allowed
  INDF1 = X;            // Illegal writing to INDF1 (see setINDF1)
  FSR0++;
  FSR1++;
}
```

Example 2

```
      // Allowed
FSR0 = ADDR_FROM;
FSR1 = ADDR_TO;
for(i = 0; i < 16; i++)
{
  X = INDF0;            // Allowed (and recommended)
  setINDF1(X);
  FSR0++;
  FSR1++;
}
```

Example 3

```
      // Also correct but more memory consuming
FSR0 = ADDR_FROM;
FSR1 = ADDR_TO;
for(i = 0; i < 16; i++)
{
  X = getINDF0();
  setINDF1(X);
  FSR0++;
  FSR1++;
}           // Result is the same as copyMemoryBlock(ADDR_FROM, ADDR_TO, 16)
```

getINDF1

Function	Read a value in the `INDF1` virtual register
Purpose	Indirect read from RAM
Syntax	`uns8` **`getINDF1()`**
Parameters	–
Return value	Value of the register addressed by the `FSR1H` and `FSR1L`
Output values	–
Preconditions	• `FSR1` (the `FSR1H` and `FSR1L` register pair) must be set before to define a destination. Traditional as well as linear address can be used.
Remarks	• See example E06–RAM [10]. • Additionally, simple reading the `INDF1` virtual register is allowed. This is even less memory consuming than `getINDF1`. See getINDF0, Example 2 . • Another possibility is using the `readFromRAM` function. • `getINDF1` can advantageously be replaced by direct reading, e.g. `X = getINDF1;` see `getINDF0`, Example 2.
Side effects	–
See also	`setINDF0`, `setINDF1`, `getINDF0`, `readFromRAM`, `copyMemoryBlock`
Example	See `getINDF0 (by analogy with INDF1)`

Buffers

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, further optionally reduced by memoryLimit). In addition to this, the memoryLimit variable can be used to specify number of bytes to be transferred.

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

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	• If memoryOffsetFrom = 0, memoryOffsetTo = 0 and memoryLimit = 0, complete 64 B is copied. • See example E06 - RAM [10].
Side effects	–
See also	clearBufferINFO, copyBufferINFO2RF, copyBufferRF2COM, copyBufferRF2INFO, copyBufferCOM2RF, copyBufferCOM2INFO, compareBufferINFO2RF, copyMemoryBlock
Example 1	`copyBufferINFO2COM();`

Example 2	`memoryOffsetFrom = 0;  // bufferINFO to be copied`
	`memoryOffsetTo = 10;  // to  bufferCOM starting from bufferCOM[10].`
	`copyBufferINFO2COM;  // Just first 54 B is copied (until bufferCOM full).`

Example 3	`memoryOffsetFrom = 0;  // bufferINFO to be copied`
	`memoryOffsetTo = 10;  // to  bufferCOM starting from bufferCOM[10].`
	`memoryLimit = 20;`
	`copyBufferINFO2COM;  // Just first 20 B is copied (due to memoryLimit).`

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	• If `memoryOffsetFrom` = `0`, `memoryOffsetTo` = `0` and `memoryLimit` = `0`, complete 64 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	Offsets `memoryOffsetFrom` and `memoryOffsetTo` are applied (see above).
Remarks	• If `memoryOffsetFrom` = `0`, `memoryOffsetTo` = `0` and `memoryLimit` = `0`, complete 64 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	Offsets `memoryOffsetFrom` and `memoryOffsetTo` are applied (see above).
Remarks	• Copying is limited up to first 64 B of `bufferRF` only. • If `memoryOffsetFrom` = `0`, `memoryOffsetTo` = `0` and `memoryLimit` = `0`, complete 64 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	Offsets `memoryOffsetFrom` and `memoryOffsetTo` are applied (see above).
Remarks	• If `memoryOffsetFrom` = `0`, `memoryOffsetTo` = `0` and `memoryLimit` = `0`, complete 64 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	Offsets `memoryOffsetFrom` and `memoryOffsetTo` are applied (see above).
Remarks	• If `memoryOffsetFrom = 0`, `memoryOffsetTo = 0` and `memoryLimit = 0`, complete 64 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 64)
Return value	• 1 – match • 0 – mismatch
Output values	–
Preconditions	–
Remarks	• Comparing is limited up to first 64 B of bufferRF only. • If `memoryOffsetFrom = 0`, `memoryOffsetTo = 0` and `memoryLimit = 0`, complete 64 B is compared. • See example E06 - RAM [10].
Side effects	–
See also	`clearBufferINFO, copyBufferINFO2RF, copyBufferRF2INFO, swapBufferINFO`
Example	`if (!compareBufferINFO2RF(32))  // Compare 32 B` `  then Error = 1;  // Error if mismatch`

swapBufferINFO

Function	Swap `bufferINFO` and `bufferAUX`
Purpose	Temporary `bufferINFO` saving
Syntax	`void` **`swapBufferINFO()`**
Parameters	–
Return value	–
Output values	Content of `bufferINFO` and `bufferAUX` (64 B) is swapped. See example E06 - RAM [10].
Preconditions	–
Remarks	–
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` (64 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` is cleared (filled with zeros). See example E06 - RAM [10].
Side effects	–
See also	`copyBufferRF2COM, copyBufferRF2INFO, copyBufferCOM2RF, copyBufferINFO2RF, compareBufferINFO2RF, copyMemoryBlock`
Example	`clearBufferRF();`

Data blocks

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	• Either traditional or linear addresses can be used. • Upward overlapping the source and the destination RAM blocks being copied is not allowed. • Avoid writing to RAM areas dedicated to OS otherwise OS can collapse. See the RAM map [2].
Remarks	• See RAM map [2] and example E06 - RAM [10].
Side effects	FSR0 and FSR1 registers are modified.
See also	`writeToRAM`, `readFromRAM`, `setINDF0`, `getINDF0`, `setINDF1`, `getINDF0`
Example 1	`copyMemoryBlock(0x2390, 0x23C0, 10); // copy 10 B block from 0x2390 to 0x23C0`
Example 2	`copyMemoryBlock(bufferRF+10, bufferCOM+1, 8);  // 8 bytes copied:` `  // bufferCOM[1] = bufferRF[10] ...  bufferCOM[8] = bufferRF[17]`
Example 3	`copyMemoryBlock(array+0, array+1, sizeof(array)-1);  // Upward, not allowed`
Example 4	`copyMemoryBlock( array+1, array+0, sizeof(array)-1 );  // Downward, allowed`

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]`:

Address in `bufferInfo`	0	1	2	3	4	5	6	7
Meaning	Serial number				OS version	TR type	OS build	
	Module ID							

Serial number (Module ID): 4 B identification code unique for each TR module.

DCTR identification:
The most significant bit of Module ID:
0: TR (fully programmable, General HWP not allowed)
1: DCTR (fully programmable, General HWP allowed)

OS version:
Upper nibble (4 b): Major version
Lower nibble (4 b): Minor version. Postfix "D" is not stated in Module identification but can be recognized by MCU type ("D" for PIC16LF1938).

TR type:

Bit	7	6	5	4	3	2	1	0
Meaning	TR series				FCC	MCU type		

TR series
0: (DC)TR-52D
1: (DC)TR-58D-RJ
2: (DC)TR-72D
3: (DC)TR-53D
8: (DC)TR-54D
9: (DC)TR-55D
10: (DC)TR-56D
11: (DC)TR-76D

FCC
0: FCC not certified
1: FCC certified

MCU type
3: PIC16F886
4: PIC16LF1938

OS build: OS subversion.

Examples (all in hexadecimal):

```
                  [0] [1] [2] [3] [4] [5] [6] [7]
bufferINFO[0-7] =  1C  10  00  01  37  24  39  11
```

Meaning: Module ID = 0100101C, TR, IQRF OS version 3.07D, TR-72D, PIC16LF1938, FCC not certified, OS build 0x1139.

```
bufferINFO[0-7] =  1C  10  00  81  37  BC  39  11
```

Meaning: Module ID = 8100101C, DCTR, IQRF OS version 3.07D, TR-76D, PIC16LF1938, FCC certified, OS build 0x1139.

Preconditions	–
Remarks	• Tip: The most significant bit in TR series can be used to differentiate between TR modules with shared and not shared MCU pins on the Cx SIM pads, e.g. TR-72D (shared) vs. TR-76D (not shared). • Module data can also be read by SPI master. See the IQRF SPI specification [5].
Side effects	`bufferINFO[8 to 63]` is modified.
See also	`appInfo`

Example 1	`uns24 SN @ bufferInfo;`
	`uns8 OSv  @ bufferInfo[4];`
	`moduleInfo();  // Now SN == module serial number`
	`// and OSv == OS version`

Example 2	`moduleInfo();`
	`if (bufferInfo[5].7 == 0)`
	`...  // MCU pins are shared (e.g. RC5 and RC7 to TR pin C8 etc.)`
	`else`
	`...  // MCU pins are not shared (e.g. RC5 and RC7 to TR pin C8 etc.)`
	`// See simplified circuit diagram in TR datasheets`

appInfo

Function	Store Application information 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`

Example 1	`appInfo();  // Copy Application info from EEPROM to bufferINFO`
	`copyBufferINFO2RF();  // and then to bufferRF`

Example 2	`#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 to C8 or Q6, Q7, Q8 and Q11) are configured and activated as SPI Slave. • See SPI Implementation in IQRF TR modules [5] and example E07-SPI [10].
Side effects	Related pins can not be used as general I/Os until SPI is disabled via `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 to C8 or Q6, Q7, Q8 and Q11) 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 64)
Return value	–
Output values	SPI Status is switched to: • *SPI data ready* – after `startSPI(1` to 64`)` • *SPI ready, Communication mode* – after `startSPI(0).`
Preconditions	• SPI must be enabled by the `enableSPI` function before. • startSPI must not be combined with functions changing bufferCOM, e.g. `discovery`
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`

Example 1	`// 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`

Example 2	`startSPI(0);  // Reset SPI communication`
Example 3	`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`
	`  //  (not to destroy bufferCOM in background)`
	`  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`

Example	`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`

Example 1

```
     // 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.
```

Example 2

```
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.
Return value	–
Output values	Available read only in the `RFpower` register
Preconditions	–
Remarks	–
Side effects	–
See also	`RFTXpacket`
Example	`setTXpower(7);  // Max. RF output power`

setRFspeed

Function	Select RF bit rate. **Not implemented yet. Do not use this function. Bit rate 19.836 kb/s is fixed.**
Purpose	Select RF bit rate
Syntax	**`void setRFspeed(speed)`**
Parameters	`uns8 speed:`
Return value	–
Output values	Available read only in the `RFspeed` register
Preconditions	Bit rates different from 19.836 kb/s are preliminary, for experimental purpose only.
Remarks	• Non-default bit rates are provisionally intended for experimental purposes only. • Routing is supported for 19.836 kb/s only
Side effects	RF channel must be specified after every bit rate change.
See also	`setRFchannel`
Example 1	`setRFspeed(1);  // 1.2 kb/s selected` `setRFchannel(...);  // channel must be selected then`
Example 2	`setRFspeed(2);  // 19.836 kb/s selected` `setRFchannel(...);  // channel must be selected then`

setRFband

Function	Select RF frequency band. **Not implemented.**
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	Flag `_916MHz` in the `userInterface` register:
	`_916MHz: 0 – 868 MHz band`
	`1 – 916 MHz band`

Preconditions	• This function is not implemented for TR modules with fixed band. • To ensure compatibility with future OS versions, it is not recommended to use this function. RF band should be configured in IQRF IDE (*TR Configuration*).
Remarks	Default RF band and channel after reset are set according to *TR Configuration*.
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
	• Default:
	  • 868 MHz band: `52`
	  Default channel can be changed by *TR Configuration* in IQRF IDE.

Return value	–
Output values	Available read only in the `RFchannel` register

Preconditions	To avoid interferences on given channel, any other RF signal from outside should differ at least for 3 in STD mode and 10 in LP or XLP mode (the more the better).
	Example:
	• In STD mode:
	  • Channels 50 and 53 typically do not interfere each other. These channels can be used in two overlapping networks (in most cases).
	• In LP or XLP mode:
	  • Channels 50 and 60 typically do not interfere each other. These channels can be used in two overlapping networks (in most cases).

Remarks	Channel 0 is reserved for DPA service purposes. It is not recommended to use it for regular communication.
Side effects	–
See also	`setRFspeed`
Example	`setRFchannel(25);  // 868.15 MHz channel selected`

setRFmode

Function	Set RF mode
Purpose	Specify RFRX and RFTX power modes, signal filtering modes for RFRX and enable fast response to external signal during LP/XLP RFRX.
Syntax	**`void setRFmode(mode)`**
Parameters	`uns8 mode: SWTTFFRR` in binary • S Ignored • W **Wait packet end** 1 Waits until receipt is finished if data payload (but not the preamble) is currently receiving, 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 ~50 ms) 10 for XLP RX mode (prolonged preamble ~900 ms) 11 LP/XLP RX asynchronous termination on pin change enabled. If enabled and no packet is just received, low level on pin C5 (for TR modules in SIM format) or Q12 (for TR-76D) terminates RF reception in LP/XLP mode, immediately or after current cycle finishing. See examples 3 and 4. • FF Not used • 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 Not used
Return value	–
Output values	Available read only in the `RFmodeByte` register.
Preconditions	• Default value is mode = 0. • Non-STD RX modes are intended for bit rate 19.386 kb/s only.
Remarks	*Tip:* As the parameters, use constants (and their ored combinations), especially the predefined ones in `IQRF-macros.h` header file instead of binary values. See example E10-RFMODE and Example 1 below.
Side effects	RF circuitry and MCU is temporarily set to sleep during low power RX modes. Thus, all tasks running on OS background (e.g. SPI communication, LED indication etc.) can be untimely canceled. To avoid this, use `setRFmode` after finishing all background tasks. See Example 2.
See also	`checkRF`

Example 1	`// RX: STD, no filtering, TX: for STD RX (standard preambles)`
	`setRFmode(0b00000000);  // Using numeral value`
	`setRFmode(_RX_STD | _TX_STD)  // The same using predefined constants for clarity`
	`// RX: LP, TX: for LP RX (prolonged preambles ~50 ms)`
	`setRFmode(0b00010001);  // Using numeral value`
	`setRFmode(_RX_LP | _TX_LP)  // The same using predefined constants for clarity`

Example 2	`while (getStatusSPI())  // Wait for finishing SPI on background`
	`  clrwdt();`
	`disableSPI();`
	`SWDTEN = 0;  // Possibly disable WDT for lower consumption`
	`setRFmode(_RX_LP | _TX_LP);  // and go to LP mode then`

Example 3

```
// RFRXpacket terminated after low level on C5/Q12 is detected and
// current cycle is finished.
toutRF = 100;                 // [in cycles], 1 cycle == ~770 ms
while (1)
{
  setRFmode(_RX_XLP | _RLPMAT);  // RX_XLP + LP/XLP RX termination
  if (RFRXpacket())
  {
    ...
  }
  …                     // Goes here after every 77 s (toutRF=100) or
                        //   if low level appears on the C5/Q12 pin
                        //      in a moment when RX XLP cycle is finished
  if (buttonPressed)
  {
    ...
    setRFmode(_RX_STD);       // Set required TX preamble
    RFTXpacket();
  }
}
```

Example 4

```
// RFRXpacket terminated immediately after low level on C5/Q12 is detected.
// It is necessary to activate interrupt on change periodically.
toutRF = 100;                       // [in cycles], 1 cycle == ~770 ms
while (1)
{
  setRFmode(_RX_XLP | _RLPMAT);     // RX_XLP + LP/XLP RX termination
  writeToRAM(&IOCBN, IOCBN | 0x10); // Negative edge active.
                                    // Instead of IOCBN.4=1;
                                    // Bit IOCBN.4 cannot be accessed
                                    // directly due to OS restriction.
  IOCBP.4 = 1;                      // Positive edge active too
  IOCIE = 1;                        // Interrupt on change enabled
  writeToRAM(&IOCBF, IOCBF & 0xEF); // Clear interrupt on change flag.
                                    // Instead of IOCBF.4=0;
                                    // Bit IOCBF.4 cannot be accessed
                                    // directly due to OS restriction.
  if (RFRXpacket())
  {
    ...
  }
  …                   // Goes here after every 77 s (toutRF=100) or
                      //   immediately if low level appears on the C5/Q12 pin
  if (buttonPressed)
  {
    ...
    setRFmode(_RX_STD);             // Set required TX preamble
    RFTXpacket();
  }
}
```

checkRF

Function	Check incoming RF signal strength for specified level.
Purpose	Incoming RF signal detection (instead of dummy `RFRXpacket`).
Syntax	**`bit checkRF(level)`**
Parameters	`uns8 level = RSSI_FILTER` `RSSI_FILTER`: 0 to 100. Values greater than 100 are truncated to 100. RSSI level in dBm can be evaluated as `RSSI = -112 + RSSI_FILTER` [dBm] • `RSSI_FILTER == 0` means signal level -112 dBm • `RSSI_FILTER == 5` means signal level -107 dBm • ... • `RSSI_FILTER == 100` means signal level -12 dBm Higher level requires stronger signal. Relative RF range is shortened due to this filtration according the datasheet of the TR module.

Return value	• `0`: Signal with specified level or higher not detected
	`RSSI < RSSI_FILTER`
	• `1`: Signal with specified level or higher detected
	`RSSI >= RSSI_FILTER`

Output values	After `checkRF` finishing, RF IC stays always in RF Ready mode.
Preconditions	• In LP and XLP RX modes, `checkRF` should be used only once whenever a change of filter level is needed. It should not be used repeteadly in RX loop.
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. • For environment without a significant noise `checkRF(0)` is recommended. • Checking takes about 1 ms. • `checkRF` performs the measurement only but does not store the result (`lastRSSI` is not updated). For reading out the value the `getRSSI` function is intended. See `getRSSI` Example.
Side effects	–
See also	`RFRXpacket`, `getRSSI`

Example 1

```
            // Fast response receiving in STD mode
if (checkRF(5))      // Detect signal with RSSI >= -107dBm
{
  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
```

Example 2

```
if (checkRF(10))     // Detect signal with RSSI >= -102dBm
   ...
```

Example 3

```
            // RF signal strength analyzer
while (1)
  if (checkRF(5)) pulseLEDR(); // LED flash if RSSI >= -107dBm detected
```

getRSSI

Function	Reads the `RSSI_LEVEL` register from RF IC. The current value is not measured but just read out the last one.
Purpose	Gets the RF signal level, especially for fast check without receiving.
Syntax	`uns8` **`getRSSI()`**
Parameters	–
Return value	`RSSI_LEVEL` value at the time of the last `checkRF` (or `RFRXpacket`) call. `RSSI [dBm] = RSSI_LEVEL – 130`. See the RF IC datasheet.
Output values	Return value is also copied to the `lastRSSI` register.
Preconditions	Return value is valid only if `checkRF` (or successful `RFRXpacket`) had been called before.
Remarks	The `lastRSSI` register is updated also automatically: • after `RFRXpacket` if returns `true`. Thus, it is not meaningful to call `getRSSI` after `RFRXpacket`. • after `getRSSI (`valid after preceding `checkRF` call)
Side effects	–
See also	`checkRF`, `RFRXpacket`
Example	`checkRF(0);` `i = getRSSI();  // Get current RSSI level`

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-DFM-C [10].
Side effects	• bufferRF[DLEN] and bufferRF[DLEN+1] are destroyed • System tick timing is slightly affected. • The RF circuitry wakes up (in case of sleeping).
See also	`RFRXpacket`, `setTXpower`, `setRFmode` and (in case of IQMESH) also other RF functions

Example 1

```
   // 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
```

Example 2

```
   // 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)
```

Example 3

```
   // 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)
RTHOPS = 10;          // 10 hops
// RTHOPS = eeReadByte[0]; // # hops = # bonded nodes

RTTSLOT = 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. RSSI [dBm] = lastRSSI – 130.
	• `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, see IQRF OS User's guide, RF RX and TX 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 non-networking 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`. • `RFRXpacket` is abandoned cca 105 ms (in LP mode) or cca 1005 ms (in XLP mode) after the packet transmission start. • In LP and XLP modes both LEDs are switched off. • After termination in LP mode, RF IC is switched to RF ready mode. • After termination in XLP mode, RF IC is switched to RF sleep mode. • `RFRXpacket` is allowed to be called at least 5 ms after `RFTXpacket` in LP and XLP modes. See Example 4. • See examples E02–RX, E03–TR, E09–LINK, E11-IQMESH-DFM-N and E14-CONSUMPTION [10].
Side effects	• Update PIN before every `RFTXpacket` followed after `RFRXpacket`. • Result of `captureTicks` is destroyed if `startCapture` is active on background at the same time. • System tick timing is slightly affected. • bufferRF[DLEN] and bufferRF[DLEN+1] is destroyed. • The RF circuitry wakes up (in case of sleeping). • If a packet received the A/D converter control registers are changed.
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
    waitNewTick();
    while (RTHOPS)         // RTHOPS - rest of hops
    {
      waitDelay(RTTSLOT); // RTTSLOT - timeslot
      RTHOPS--;           // Do not answer until all hops are finished
    }
  }
  ...                      // Now the Node is allowed to answer
}
```

Example 4

```
setRFmode(_RX_LP);        // or _RX_XLP
  ...
RFTXpacket();
  ...                     // If there is no other code taking at least 5 ms,
waitMS(5);                //    the delay must be inluded here
if (RFRXpacket())
  ...
```

Networking

setCoordinatorMode

Function	Set Coordinator mode
Purpose	Assign the TR module as a network Coordinator
Syntax	`void` **`setCoordinatorMode()`**
Parameters	–
Return value	–
Output values	• Flag `_networkingMode` (`userInterface.7`) = 1 • Flag `_networkTwo (userInterface.6) = 0` • In Coordinator mode the `_NTWF` flag (`PIN.7`) is automatically set before calling `RFTXpacket`
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, RFTXpacket`
Example	

setNodeMode

Function	Set Node mode
Purpose	Assign the TR module as a network Node
Syntax	`void` **`setNodeMode()`**
Parameters	–
Return value	–
Output values	• Flag `_networkingMode (userInterface.7) = 1` • Flag `_networkTwo (userInterface.6) = 1` • In Node mode the `_NTWF` flag (`PIN.7`) is automatically set before calling `RFTXpacket`
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, RFTXpacket`
Example	

setNonetMode

Function	Select Peer-to-peer mode
Purpose	Switch from IQMESH to Peer-to-peer
Syntax	`void` **`setNonetMode()`**
Parameters	–
Return value	–
Output values	• Flag `_networkingMode` (`userInterface.7`) = `0`
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`. • Flag `_networkTwo (userInterface.6)` is not changed.
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	Flag `_filterCurrentNetwork` in register `userInterface`:
	`_filterCurrentNetwork: 0 – filtering off`
	`1 – filtering on`
	• 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	• Flag `_filterCurrentNetwork` in register `userInterface`:
	`_filterCurrentNetwork: 0 – filtering off`
	`1 – filtering on`
	• 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 procedure by the user (to keep user program the same for all Nodes etc.). • Node User address is stored in EEPROM and is accessible via `getNetworkParams`. See Example 4.
Side effects	–
See also	`bondNewNode`
Example 1	`setUserAddress(2000);  // The Node has got user address 2000`

Example 2	`setUserAddress(UA);`
	`eeWriteByte(EEUA, UA)  // User address stored to EEPROM`
	`...`
	`reset();  // User address lost after reset`
	`setUserAddress(eeReadByte(EEUA));  // User address restored from EEPROM`

Example 3	`getNetworkParams();  // Get User address`
	`uns16 myAddress = ntwUSERADDRESS;  // See IQRF-memory.h`

getNetworkParams

Function	Get network parameters
Purpose	Get some information about curent system, RF and network parameters
Syntax	**`getNetworkParams()`**
Parameters	–
Return value	–

Output values	• `param2`: Address of the device in network.
	  • 0 Unbonded device
	  • 1 – 239 Bonded Node (logical address)
	  • 254 (0xFE) Universal address (e.g. prebonded Node)
	• `bit _NTWPACKET`
	  1 – IQMESH packet
	  0 – Peer-to-peer packet
	• `param3`: Network identification (`param3.high=NID1`, `param3.low=NID0`).
	  If the device is bonded `NID0` and `NID1` refer to Coordinator otherwise to the device itself. These features are not guaranteed for future OS versions.
	• Network parameters (registers with names beginning with the `ntw` prefix) are updated. See IQRF OS User's guide, Appendix 2, table OS, RF and network parameters.

Preconditions	For IQMESH only.
Remarks	See example E11 - IQMESH-DFM-N [10].
Side effects	–
See also	`amIBonded, removeBondAddress`

Example	`if (amIBonded())  // Is the Node bonded?`
	`{  // Yes:`
	`  getNetworkParams();  //  Get Node number`
	`  myAddr = param2;`
	`}`

sendFRC

Function	Fast Response Command (FRC) by the Coordinator and receiving of fast answer from all Nodes
Purpose	Send a requesting command and receive fast answer with data collection from more Nodes
Syntax	`uns8` **`sendFRC(command)`**

Parameters	`uns8 command`: User command. It is copied to `MPRW1` on Node side.
	• command.7 Format of collected data
	  • 0 Bit pairs collected. 2 bits from up to 239 Nodes (with logical addresses 1-239)
	  • 1 Bytes collected:
	  • 1B mode: 1 byte from up to 62 Nodes:
	  • For not selective FRC: from nodes with logical addresses 1-62
	  • For selective FRC: from up to 62 nodes selected from 239 Nodes
	  • 2B mode: 2 bytes from up to 30 Nodes:
	  • For not selective FRC: from nodes with logical addresses 1-30
	  • For selective FRC: from up to 30 nodes selected from 239 Nodes
	• command.0 to .6 User-specific (possibly closer specifying the FRC command)

Return value	• 0x00 – 0xEF FRC successful. Number of Nodes participating in FRC (adding values to FRC response). For bit pairs collected only. Just non-zero bit pairs are counted. • 0xF0 – 0xFC Reserved • 0xFD FRC unsuccessful. Immediate return: max. number of selected Nodes (specified in bit array) allowed for selective FRC exceeded (>62 b for 1B FRC or >30 b for 2B FRC). • 0xFE FRC unsuccessful. Immediate return in case of EEPROM non-consistency (e.g. not initialized EEPROM by `clearAllBonds` before new bonding). For bit pairs collected only. • 0xFF FRC unsuccessful, no Nodes are bonded

Output values	• Collected data is stored in `bufferINFO` (if properly answered by the Nodes)
	  • When bits pairs are collected, the 1st bits from the Nodes are stored in the bytes indexed 0-29 of the `bufferINFO`, 2nd bits from the Nodes are stored in the bytes indexed 32-61. Bit.0 in `bufferINFO[0]` and `bufferINFO`[32] is not used.
	`bufferINFO [0]` `bufferINFO[1] ...`
	7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
	1st bit of: N7 N6 N5 N4 N3 N2 N1 - N15 N14 N13 N12 N11 N10 N9 N8
	`... bufferINFO [32]` `bufferINFO[33] ...`
	7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
	2nd bit of: N7 N6 N5 N4 N3 N2 N1 - N15 N14 N13 N12 N11 N10 N9 N8
	For selective FRC, only values corresponding to selected Nodes are valid.
	  • In 1B mode, collected data is stored at bytes 1-62 of the `bufferINFO`. `bufferINFO[0]` is not used.
	bufferINFO [0] [1] [2] [3] [4] …
	- N1 N2 N3 N4 ... For non-selective FRC.
	- S1 S2 S3 S4 ... For selective FRC. S1 … S62 mean up to 62 selected Nodes (selected from N1 to N239 by the bit array, see *Preconditions*).
	  • In 2B mode, collected data (little endian) is stored at bytes 2-61 of the `bufferINFO`. `bufferINFO[0]` and `[1]` are not used.
	BufferINFO [0] [1] [2] [3] [4] [5] …
	- - N1 N2 … For non-selective FRC.
	- - S1 S2 … For selective FRC. S1 … S30 mean up to 30 selected Nodes (selected from N1 to N239 by the bit array, see *Preconditions*).

Preconditions	• The 2 B in Standard FRC or 30 B in Advanced FRC array `DataInSendFRC` of the Coordinator should be specified. This array will be copied to the `DataOutBeforeResponseFRC` array of all Nodes which received FRC.
	• `bufferINFO[0-29]`: For selective FRC only. The bit array specifying (by log. 1) selected Nodes in following order:
	`bufferINFO[0].0` – unused, `bufferINFO[0].1` – N1, ..., `bufferINFO[0].7` – N7,
	`bufferINFO[1].0` – N8, `bufferINFO[1].1` – N9, ..., `bufferINFO[29].7` – N239
	• For IQMESH Coordinator only. (`setCoordinatorMode` is automatically called first).
	• FRC modes must always be selected:
	  • By bit `_advancedFRCmode`:
	  • 0: Standard FRC mode
	  • 1: Advanced FRC mode
	  • By bit `_selectiveFRCmode`:
	  • 0: Non-selective FRC
	  • 1: Selective FRC
	  • By bit `_twoByteFRCmode`:
	  • 0: 2 b or 1 B response is requested. See *Parameters*.
	  • 1: 2 B response is requested. See *Parameters*.
	These selection bits are undefined after reset and always left completely under user control (not affected by OS).
	• The time needed to complete FRC responses in Nodes must be specified by the Coordinator by macro `setResponseFRCtime`. It is the maximal time between finished routing and `responseFRC` calls, the same for all responsing Nodes. It is selectable from 8 possible values (from 40 ms to 20.48 s) and passed to responsing Nodes within the `sendFRC` command. See `responseFRC`, *Example 1*. Header file `IQRF-macros.h` defines all the 8 possible periods by constants `_RESPONSE_FRC_TIME_xxx_MS`. The time required to complete the FRC answers depends on the application and must be selected according to the Node needed the longest time to acquire response data. For example, if the Node needs up to 400 ms to complete data from a sensor, the nearests higher time `_RESPONSE_FRC_TIME_640_MS` should be used.
	• FRC works for 1 B addressing (with addresses from 1 up to 239) only.
	• `clearBufferINFO()` and `PIN = 0` must be performed first. See Examples.

Remarks	• See example E11-IQMESH-DFM-C [10] and IQRF OS User's guide [1], chapter Fast Response Command.
	• Data can be collected also from not discovered Nodes.
	• This is a blocking function (application program is staying here until the collection is completed). This time depends on number of bonded and discovered Nodes. Typical time for `sendFRC` is lower than:
	  • Standard FRC:
	  `BONDED_NODES * 130 + _RESPONSE_FRC_TIME_xxx_MS + 250` [ms]
	  • Advanced FRC and STD mode:
	  `BONDED_NODES * 150 + _RESPONSE_FRC_TIME_xxx_MS + 290` [ms]
	  • Advanced FRC and LP mode:
	  `BONDED_NODES * 200 + _RESPONSE_FRC_TIME_xxx_MS + 390` [ms]
	• Standard FRC works in RF STD mode only. Advanced FRC works in RF STD or LP modes only.

Side effects	• OS buffers (`bufferINFO`, `bufferRF` and `bufferAUX`) are modified • All OS registers regarding RF communication – relating to network parameters sent in the packet, e.g. `RX` and `RTDT0` – `RTDT3` (`RTHOPS`, `RTTSLOT`, …) may be changed. • A/D converter control registers are changed
See also	`ResponseFRC, amIRecipientOfFRC`

Example 1

```
// Standard, Non-selective, 2 bits collecting, STD RF mode
PIN = 0;
clearBufferINFO();
_advancedFRCmode = 0;               // Standard FRC
_selectiveFRCmode = 0;              // Non-selective FRC
_twoByteFRCmode = 0;                // Non-Two byte FRC
setResponseFRCtime(_RESPONSE_FRC_TIME_40_MS);  // responseFRC must be called
                                // up to 40 ms after sendFRC routing finishing
DataInSendFRC[0] = user_value0;     // 2 B data to be delivered to all Nodes
DataInSendFRC[1] = user_value1;
stopSPI();
_LEDG = 1;                          // FRC duration indication
SendFRC(myCommand & 0x7F);          // Bit 7 must be cleared to collect bits
_LEDG = 0;
copyBufferINFO2COM();
startSPI(sizeof(bufferCOM));
```

Example 2

```
// Advanced, Non-selective, 1 byte collecting, LP RF mode
PIN = 0;
clearBufferINFO();
_advancedFRCmode = 1;               // Advanced FRC
_selectiveFRCmode = 0;              // Non-selective FRC
_twoByteFRCmode = 0;                // Non-Two byte FRC
setResponseFRCtime(_RESPONSE_FRC_TIME_320_MS);  // responseFRC must be called
                                // up to 320 ms after sendFRC routing finishing
setRFmode(_TX_LP);                  // Can work also in LP TX mode
DataInSendFRC[0] = user_value0;     // 30 B data to be delivered to all Nodes
      ...
DataInSendFRC[29] = user_value29;
stopSPI();
_LEDG = 1;                          // FRC duration indication
SendFRC(myCommand | 0x80);         // Bit 7 must be set to 1 to collect bytes
_LEDG = 0;
copyBufferINFO2COM();
startSPI(sizeof(bufferCOM));
```

Example 3

```
// Advanced, Selective, 2 bits collecting, LP RF mode
PIN = 0;
clearBufferINFO();
_advancedFRCmode = 1;               // Advanced FRC
_selectiveFRCmode = 1;              // Selective FRC
_twoByteFRCmode = 0;                // Non-Two byte FRC
setResponseFRCtime(_RESPONSE_FRC_TIME_640_MS); // responseFRC must be called
                        //up to 640 ms after sendFRC routing finishing
bufferINFO[0] = 0x0A;   // Set bit field of selected Nodes in bufferINFO
// bufferINFO[1] = xx;              //   N1 and N3 are selected in this case
//    ...
setRFmode(_TX_LP);                  // Can work also in LP TX mode
DataInSendFRC[0] = user_value0;     // 30 B data to be delivered to all Nodes
      ...
DataInSendFRC[29] = user_value29;
stopSPI();
_LEDG = 1;                          // FRC duration indication
SendFRC(myCommand & 0x7F);          // Bit 7 must be cleared to collect bits
_LEDG = 0;
copyBufferINFO2COM();
startSPI(sizeof(bufferCOM));
```

Example 4

```
// Advanced, Selective, 2 bytes collecting, LP RF mode
PIN = 0;
clearBufferINFO();
_advancedFRCmode = 1;               // Advanced FRC
_selectiveFRCmode = 1;              // Selective FRC
_twoByteFRCmode = 1;                // Two byte FRC
setResponseFRCtime(_RESPONSE_FRC_TIME_640_MS); // responseFRC must be called
                                 // up to 640 ms after sendFRC routing finishing
bufferINFO[0] = 0x0A;            // Set bit field of selected Nodes in bufferINFO
// bufferINFO[1] = xx;              // N1 and N3 are selected in this case
//    ...
setRFmode(_TX_LP);                  // Can work also in LP TX mode
DataInSendFRC[0] = user_value0;     // 30 B data to be delivered to all Nodes
      ...
DataInSendFRC[29] = user_value29;
stopSPI();
_LEDG = 1;                          // FRC duration indication
SendFRC(myCommand | 0x80);          // Bit 7 must be 1 to collect bytes
_LEDG = 0;
copyBufferINFO2COM();
startSPI(sizeof(bufferCOM));
```

responseFRC

Function	Response to FRC command received by a Node
Purpose	Fast sending of collected data from more Nodes to the Coordinator
Syntax	`void` **`responseFRC()`**
Parameters	–
Return value	–
Output values	Requested data is delivered to the Coordinator

Preconditions	• FRC packet received is indicated by the `_wasFRC` flag.
	• As a result of preceding FRC command received, the following variables are set:
	  • `MPRW0` contains the `__FRCOMMAND` (standard FRC) or `__FRCOMMANDADV` (Advanced FRC) constant value.
	  • `MPRW1` contains a user command sent from the Coordinator as the parameter of function `sendFRC`
	  • Bit.7 in register `MPRW1` specifies the format (the range and the type) of the response data to be collected:
	  • 0 2 bits (`responseFRCvalue.0` and `.1`) – from all Nodes with logical addresses 1-239
	  • 1 Bytes:
	  Flag `_twoByteFRC` 0:
	  1 byte (`responseFRCvalue`) – from up to 62 selected Nodes:
	  • For non-selective FRC: from Nodes with addresses 1-62
	  • For selective FRC: from up to 62 selected Nodes (selected from N1–N239 in bit array)
	  Flag `_twoByteFRC` 1:
	  2 bytes (`responseFRCvalue2B`) – from up to 30 selected Nodes:
	  • For non-selective FRC: from Nodes with addresses 1-30
	  • For selective FRC: from up to 30 selected Nodes (selected from N1–N239 in bit array)
	  • `param4` contains the time (in ticks) specified on the Coordinator side by macro `setResponseFRCtime`. See below.
	• Before `responseFRC` calling, it is necessary to wait until routing of the FRC packet from the Coordinator is finished. See the example below.
	• Maximal time between finished routing and responseFRC calling (i.e. the time for handling the data for FRC answer) must be the same for all Nodes and is specified by the Coordinator (by macro `setResponseFRCtime`). See `sendFRC` Examples. This time is propagated throughout the network and placed to registers `param4` in Nodes. This is intended to generate proper delay ensuring correct timing of FRC respones by startLongDelay. See Example.
	• Before `responseFRC` calling, the response value(s) (data to collect) must be placed in the `responseFRCvalue` register or (for Two byte mode) in the `responseFRCvalue2B variable`. It is recommended to respond by non-zero values only and dedicate zero value to distinguish (by the Coordinator) the case that the response from the Node failed. (But also any other rule can be defined by the user for it instead.)
	• For IQMESH Node only.
	• For 1 B addressing (for Nodes with addresses from 1 up to 239) only.
	• It is not intended for prebonded (not authorized) Nodes. Such Nodes are ignored by `responseFRC`.
	• Standard FRC works in RF STD mode only. Advanced FRC works in RF STD or LP modes.

Remarks	• See example E11-IQMESH-DFM-N [10] and IQRF OS User's guide [1], chapter Fast Response Command.
	• This is a blocking function (application program is staying here until the collection is completed). The time depends on:
	  • Number of Nodes in the network
	  • Whether the Node is discovered or not
	  • Logical address or VRN

Side effects	• OS buffers `bufferINFO`, and `bufferRF` are modified • All OS registers regarding RF communication – relating to network parameters sent in the packet, e.g. `RX` and `RTDT0` – `RTDT3` (`RTHOPS`, `RTTSLOT`, …) may be changed. • A/D converter control registers are changed
See also	`sendFRC, amIRecipientOfFRC`

Example

```
  // Response to standard or advanced FRC, either selective or non-selective
if (RFRXpacket())
{
   if (_ROUTEF)           // Has the packet been routed?
   {                      // Yes - wait until routing is finished
      waitNewTick();
      while (RTHOPS)      // Rest of hops
      {
         waitDelay(RTTSLOT);
         RTHOPS--;
      }
    }
                          // FRC command handling
    if (_wasFRC)
    {               // FRC packet detected. Register param4 contains
                    // the time needed for FRC handlig set by
                    // setResponseFRCtime macro on the Coordinator side.
      startLongDelay(param4);
      bit FRChandled = FALSE;

      do
      {        // If the Node is a recepient of FRC, handle it only once.
        if (amIRecipientOfFRC() && (FRChandled == FALSE))
        {
          FRChandled = TRUE;
          if (MPRW0 == __FRCOMMAND)
          {
             uns16 user_value = DataOutBeforeResponseFRC;
                    // A value received from the Coordinator
          }         // (from register DataInSendFRC)
          else
          {
            uns8 user_buf[30];
            user_buf[0] = DataOutBeforeResponseFRC[0];    // Values received
                ...         // from Coordinator (from array DataInSendFRC)
            user_buf[29] = DataOutBeforeResponseFRC[29];
          }

          ...  // Do something according to MPRW1 command (and possibly
               // according to DataOutBeforeResponseFRC), e.g. myResponse=…;
               // Shaded part must take up to the time specified in param4
          if (MPRW1.7)
          {
            if (_twoByteFRC == 1)
            {                                             // 2 byte value
              responseFRCvalue2B.high8 = myResponse_HB;
              responseFRCvalue2B.low8 = myResponse_LB;
            }
            else
              responseFRCvalue = myResponse;              // 1 byte value
          }
          else
            responseFRCvalue = myResponse & 0x03;         // 2 bit value
        }
      } while (isDelay());      // Wait for rest of the time set by
                              // setResponseFRCtime macro on the Coordinator side

      responseFRC();       // Blocking time - see Remarks
   }
   else                    // Non FRC command handling
   {
      ...
   }
}
```

amIRecipientOfFRC

Function	Evaluate whether the FRC conmmand is intended for given Node
Purpose	Enable FRC response for requested Nodes only
Syntax	`bit` **`amIRecipientfOfFRC()`**
Parameters	–
Return value	• 0 FRC is not intended for given Node • 1 FRC is intended for given Node (also for non-selective FRC)
Output values	–
Preconditions	For IQMESH Nodes only.
Remarks	• `amIRecipientOfFRC` must be called after FRC command receipt but before `bufferRF` is affected later on either by OS or by the user. E.g., it must be called before `responeFRC`. • See example E11 - IQMESH-DFM-N [10].
Side effects	–
See also	`responseFRC`
Example	See `responseFRC` Example.

Routing

setRoutingOn

Function	Routing enabled
Purpose	Allow the Node to route packets on background.
Syntax	`void` **`setRoutingOn()`**
Parameters	–
Return value	–
Output values	• Enables to assign a VRN (Virtual Routing Number) to the Node during Discovery • Flag `_disableRouting = 0` • This state is stored in EEPROM and initialized after reset.
Preconditions	• For IQMESH Nodes only • For DFM routing algorithms only
Remarks	• Routing must be enabled for a Node to be assigned to the routing backbone during Discovery. • For DFM topologies, `discovery` must be called after every `setRoutingOn` otherwise the Node will not work as a router. • Routing can be enabled in STD and LP receive modes only. Routing in XLP mode is not supported for TR-7xD transceivers. • Flag `_disableRouting` in register `_ntwCFG` is available read only after calling `getNetworkParams`: `_disabledRouting: 0` – Routing on `1` – Routing off
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	• Disables to assign a VRN (Virtual Routing Number) to the Node during Discovery • Flag `_disableRouting = 1` • This state is stored in EEPROM and initialized after reset.
Preconditions	• For IQMESH Nodes only • For DFM routing algorithms only
Remarks	• If routing is disabled the Node will not be assigned to the routing backbone during Discovery. • For DFM topologies, to fix the discontinuity in the network, `discovery` must be called after every `setRoutingOff` applied on an already discovered Node. • Flag `_disableRouting` in register `_ntwCFG` is available read only after calling `getNetworkParams`: `_disabledRouting: 0` – Routing on `1` – Routing off
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(MaxNodeAddress)`**
Parameters	`uns8: MaxNodeAddress`: Maximum address of the node to be participating in the discovery process. • 1 to 239 Specified value is applied • 0 Number of bonded Nodes is applied
Return value	• Number of discovered Nodes (≤ number of Nodes specified as routers (by `setRoutingOn`)) • 0xFE – EEPROM non-consistency (e.g. not initialized EEPROM by `clearAllBonds` before new bonding). Immediate return. • 0xFC – Serial EEPROM access error. Immediate return. Additionally, the `_eeeError` flag is set in this case.

Output values	• Routing address area (1 – 239) is split into two parts:
	  • Devices with addresses from 1 to `MaxNodeAddress` will be part of the discovery process, that is why they will become routers
	  • Devices with addresses from `MaxNodeAddress+1` to 239 will not be routers. See IQRF OS guide for more information.
	• Routing backbone is stored in EEPROM

Preconditions	• For IQMESH Coordinator only. • The Coordinator must be in STD or LP TX mode. LP discovery is performed if Coordinator is in LP TX. • Nodes must be in the `answerSystemPacket` loop routine during Discovery. • LED indication of passing discovery can be disabled (to save power consumption) or enabled (for development, service or demonstration) by the `_systemLEDindication` bit variable. Default is disabled (`_systemLEDindication=0`). • To avoid a collision in `bufferCOM` (see Side effects), SPI must not run in background during `discovery`. See Example 1.
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 during the discovery process. • The Coordinator is in STD RX mode during either LP or non-LP discovery. • It is recommended to run `discovery` under stronger conditions than ones that will be used in normal communication. It can be achieved by lower RF power. • See example E11-IQMESH-DFM-C [10]. • See IQRF OS User's guide, routing algorithms.
Side effects	• All OS buffers (`bufferINFO`, `bufferCOM`, `bufferRF` and `bufferAUX`) are modified. • All OS registers regarding RF communication – relating to network parameters sent in the packet, e.g. `RX` and `RTDT0` – `RTDT3` (`RTHOPS`, `RTTSLOT`, …) may be changed. • A/D converter control registers are changed
See also	`setRoutingOn`, `setRoutingOff`, `isDiscovederedNode`, `bondNewNode`, `answerSystemPacket`

Example 1	`setTXpower(DISCOVERY_POWER);  // Set RF power for discovery`
	`setRFmode(_RX_STD);  // Set STD RX mode`
	`stopSPI();  // SPI should be stoped (only if used)`
	`nodes = discovery(10);  // Routers with addresses 1 to 10`
	`startSPI(0);  // Allow SPI communication (only if used)`
	`setRFmode(MY_RFMODE);  // Restore RF mode parameters`
	`setTXpower(MY_POWER);  // Restore RF power`

Example 2	`nodes = discovery(eeReadByte(0x00));  // Limit to number of bonded Nodes`

answerSystemPacket

Function	Enable response to Coordinator for Discovery and Node authorization
Purpose	Discovery and Node authorization (during remote bonding) support from the Node's side
Syntax	`void` **`answerSystemPacket()`**
Parameters	–
Return value	–
Output values	System information exchanged between Coordinator and the Node via system packets.
Preconditions	• For IQMESH Node only. • Must be performed in STD RX or LP RX modes only. • In LP RX mode the `_ignoreForcedRoutingLP` bit must be cleared. • Nodes must be in the `answerSystemPacket` loop routine when Discovery or Node authorization is running.
Remarks	• Nodes use the TX output power currently set in Coordinator for discovery. • It is recommended to run `discovery` under stronger conditions than ones that will be used in normal communication. It can be achieved by lower RF power.
Side effects	• TX power can be affected during Discovery process • All OS registers regarding RF communication – relating to network parameters sent in the packet, e.g. `RX` and `RTDT0` – `RTDT3` (`RTHOPS`, `RTTSLOT`, …) may be changed. • A/D converter control registers are modified.
See also	`setRoutingOn`, `setRoutingOff`, `isDiscovederedNode`, `discovery`, `nodeAuthorization`

Example	`toutRF = MY_TOUT_RF;`
	`if (RFRXpacket())`
	`{`
	`  …`
	`}`
	`else`
	`{`
	`  answerSystemPacket();  // To enable receiving of system packets`
	`  setTXpower(MY_POWER);  // Restore`
	`}`

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-DFM-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-DFM-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 given topology, without flooding
Syntax	`bit` **`optimizeHops(method)`**
Parameters	`uns8 method`: optimizing method • 0xFF DOM – Discovered optimized MESH: sets `RTHOPS` to `VRN` of addressed Node • 0x00 DRM – Discovered reduced MESH: sets `RTHOPS` to VRN of the first Node in the zone of the addressed Node.

Return value	• 1 – No error
	• 0 – Error
	  • **`optimizeHops`** has been called in the Node mode
	  • A discovered Node has been addressed and an external EEPROM access error occurred. Additionally, the `_eeeError` flag is set in this case.

Output values	If the addressed Node is discovered, `RTHOPS` (number of hops) is optimized otherwise `RTHOPS` is set to number of bonded Nodes.
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 = …`).
Remarks	• • See E11-IQMESH-DFM-C [10] and IQRF OS User's guide.
Side effects	–
See also	`discovery, isDiscovederedNode`

Example	`setCoordinatorMode();`
	`RX = MY_NODE;`
	`optimizeHops(0xFF);  // Modifies RTHOPS (number of hops)`
	`  .`
	`  .`
	`RFTXpacket();`

Bonding – Node only

bondRequestAdvanced

Function	Ask Coordinator for bonding or other Node or Coordinator for prebonding to the network via RF. Bond the Node in cooperation with the Coordinator or prebond the Node in cooperation with an already bonded Node or the Coordinator and record it to EEPROM. See IQRF User's guide, chapter *Bonding* for more information.
Purpose	Request by the Node to be included to the network on both Coordinator's and Node's sides. Moreover, a 2 B user data is exchanged between prebonded device and the device providing prebonding.
Syntax	`bit` **`bondRequestAdvanced()`**
Parameters	`uns16 DataInBondRequestAdvanced` User data passed to the device providing prebonding (to be stored in the `DataOutPrebondNode` variable). For prebonding only.
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 `bondRequestAdvanced` not beeing unbonded by `removeBond`. • Every `bondRequestAdvanced` calling increments the value of the internal counter (it is sent with the request). This counter is used with the `bondingMask` register to handle the situation when more than one Node with enabled prebonding would response to the request. See functions `bondNewNode` and `prebondNode` and IQRF User's guide, chapter *Bonding*. The counter is cleared after reset. • `uns16 DataOutBondRequestAdvanced` – User data passed from device providing prebonding (parameter `DataInPrebondNode` of function `prebondNode`). For prebonding only.
Preconditions	• For IQMESH Node only (`setNodeMode` must be called first)
Remarks	• Bonding is a mutual relationship between Coordinator and Node. Coordinator assigns a Node number (1 to 239 or 0xFE) 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 bonding functions. See example E11-IQMESH-DFM-N, E11-IQMESH-DFM-C [10]. • Prebonding is an initial phase of remote bonding. The new (bond requesting) Node gets the network ID and the universal address 0xFE from another (already bonded) Node or the Coordinator. So the new Node becomes accessible in given network and can be authorized by the Coordinator to get a requested address. • This function takes cca 60 ms. It sends just one request for bonding and then waits for some time for the confirmation. This is the main difference between the recent `bondRequest` function which sends the request repeatedly for cca 10 s. • Requesting packet is sent in currently selected RF TX mode (STD or LP). • RF power and RF channel is not affected. • The assigned address can be found out by function `getNetworkParams`.
Side effects	• `DLEN`, `PIN`, `bufferRF` and `bufferINFO` are modified. • IQMESH mode must be restored by `setNodeMode` after `bondRequestAdvanced`. • A/D converter control registers are modified.
See also	`bondNewNode`, `amIBonded`, `removeBond`, `rebondNode`, `getNetworkParams`, `setNodeMode`, `prebondNode`, `nodeAuthorization`, `setRFmode`

Example	`while (!amIBonded())  // Request for beeing bonded (if not bonded yet)`
	`{`
	`  setRFmode(_TX_STD)  // or setRFmode(_TX_LP) to select STD or LP bonding`
	`  clrwdt();  // If WDT active`
	`  pulseLEDG();`
	`  DataInBondRequestAdvanced = user_in_value;  // uns16, to be delivered to`
	`  // the device that provides prebonding`
	`  // (to register DataOutPrebondNode)`
	`  if (bondRequestAdvanced()) // Repeatedly try to bond`
	`  {`
	`  pulseLEDR();`
	`  uns16 user_out_value = DataOutBondRequestAdvanced;  // A value received`
	`  // from the device that provides prebonding`
	`  // (from register DataInPrebondNode)`
	`  }`
	`  waitDelay(1);`
	`}  // Until successful`
	`setNodeMode();  // Restore`

bondRequest - obsolete

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`. • Coordinator is not affected at all. • `param2`: Node address (if successfully bonded only). Not guarranted for future OS versions.
Preconditions	• Do not use this obsolete function. • For IQMESH Node only (`setNodeMode` must be called first)
Remarks	• Bonding is a mutual relationship between Coordinator and Node. Coordinator assigns a Node number (1 to 239 or 0xFE) 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-DFM-N, E11 - IQMESH-DFM-C [10]. This function is active until successfully finished or fixed 10 s timeout expired. RF power is not affected.
Side effects	• `DLEN`, `PIN`, `toutRF`, `bufferRF` and `bufferINFO` are modified • 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` • A/D converter control registers are modified
See also	`bondNewNode, amIBonded, removeBond, rebondNode, getNetworkParams, setNodeMode`

Example 1	`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();`

Example 2	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 `bondRequestAdvanced`, not beeing unbonded by `removeBond`)
	• 0 – Node is not bonded:
	  • no `bondRequestAdvanced` has ever been successfully executed
	  • after `removeBond`

Output values	–
Preconditions	For IQMESH Node only (`setNodeMode` must be called first) .
Remarks	See example E11-IQMESH-DFM-N [10].
Side effects	–
See also	`bondRequestAdvanced, removeBond, removeBondAddress`

Example	`while (!amIBonded())  // Request for beeing bonded (if not bonded yet)`
	`{`
	`  bondRequestAdvanced();  // Repeatedly try to bond`
	`  clrwdt();  // If WDT active`
	`}  // Until successful`

removeBondAddress

Function	Change logical Node address to the universal one (`0xFE`). NID stays unchanged, therefore the Node still stays in the network.
Purpose	E.g. to enable subsequent change of the Node address.
Syntax	`void` **`removeBondAddress()`**
Parameters	–
Return value	-
Output values	–
Preconditions	For IQMESH Node only (`setNodeMode` must be called first).
Remarks	• `removeBondAddress` relates to the Node only. The other side is not informed by OS about changes made by these function. If synchronization is needed it should be done by the application. • To enable possible addressing of the Node in the group of Nodes with universal address, it is recommended to save the MID of given Node by the application first.
Side effects	–
See also	`amIBonded, getNetworkParams, nodeAthorization`
Example	`removeBondAddress();  // Change current logical address to universal address`

removeBond

Function	Remove the Node from the network and record it to EEPROM.
Purpose	Exclude the Node from the network on Node's side.
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 `bondRequestAdvanced`. • Coordinator is not affected at all.
Preconditions	For IQMESH Node only (`setNodeMode` must be called first).
Remarks	• See example E11-IQMESH-DFM-N [10]. • For rebonding use bondRequestAdvanced again. • `removeBond` relates to the 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	–
See also	`bondRequestAdvanced`, `bondNewNode`, `amIBonded`, `rebondNode`
Example	`removeBond(); // Remove the bond`

Bonding – Coordinator only

bondNewNode

Function	Local bonding. Looking for bond requesting devices and bond a new Node by Coordinator on a Node's request via RF in direct range. Allocate the Node number and assign the Network ID and send both to the 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 the Coordinator in EEPROM.
Purpose	Include a new Node in direct range 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 parts of routing backbone. • 0 The first free address is assigned. It equals to the 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 only. • 254 (0xFE) The universal address. Nodes with this address are included in the network but outside the routing backbone (not being discovered). Requested address can be assigned by `setUserAddress`. It is intended especially for networks with more than 239 Nodes.

Return value	• 1 – bonding successful, the Node is included to the list of bonded Nodes.
	• 0 – bonding unsuccessful, the Node is not included to the list of bonded Nodes, immediate return:
	  • EEPROM non-consistency (e.g. not initialized EEPROM by `clearAllBonds` before new bonding).
	  • Serial EEPROM access error. Additionally, the `_eeeError` flag is set in this case.

Output values	• `param2`: Node number • `bufferRF[0 to 1]`: two lower ID bytes of the bonded Node, 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. • The Coordinator accomplishes bonding on request from a Node via RF. When this function is executing the `bondRequestAdvanced` function must be called in the Node. • It is recommended to use `bondNewMode` for bonding in STD mode only. For bonding in power saving LP mode, use the same procedure as for remote bonding (using `NodeAuthorization`) instead.
Remarks	• See example E11-IQMESH-DFM-C [10] and IQRF OS User's guide, chapter Bonding. • If no requesting Node is detected during 10 s period this function terminates and returns 0. • Network ID is derived from Coordinator ID which ensures unique identification of various networks. • RF power and RF channel is not affected. The STD mode is used. • An occupied address can be unblocked by `removeBondedNode(address)`. • The function responds to bond request only if the following internal condition is valid: `if (((counter ^ address) & bondingMask) == 0)` See `prebondNode` Remarks for description.
Side effects	The following values are modified and not restored: • `PIN`, `DLEN`, `bufferRF` and `bufferINFO` are modified • Result of `captureTicks` is destroyed if `startCapture` is active on background at the same time. • IQMESH mode must be restored by `setCoordinatorMode` after `bondNewNode` • A/D converter control registers are modified
See also	`bondRequestAdvanced`, `removeBondedNode`, `rebondNode`, `isBondedNode`, `setUserAddress`, `setCoordinatorMode`, `nodeAuthorization`, `prebondNode`

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

nodeAuthorization

Function	Authorize the prebonded Node. Allocate the Node number and send it to the prebonded Node via RF network. 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 (prebonded) Node to the network. A last phase of the remote bonding process.
Syntax	`bit` **`nodeAuthorization(mid, address)`**
Parameters	`uns16 mid:` 2 lower bytes of MID of prebonded Node to be authorized (HB = `mid[0]`, LB = `mid[1]`). `uns8 address:` • 1 to 239 Assign requested address to the prebonded Node. This must be unique in the whole network. If an existing number is used the Node is not authorized and the function immediately returns 0. Only these Nodes can be parts of routing backbone. • 0 The first free address is assigned. It equals to the 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). Requested address can be assigned by `setUserAddress`. It is intended especially for networks with more than 239 Nodes.

Return value	• 1 – The requested address was free. The Node is included into the list of bonded Nodes. The result must be verified by application program, e.g. by subsequent sending a packet from the Coordinator to the authorized Node. If the authorization failed, given address must be released by `removeBondedNode` and the authorization should be repeated.
	• 0 – Authorization unsuccessful. The Node is not included into the list of bonded Nodes. Immediate return.
	  • The requested address is already used.
	  • EEPROM non-consistency (e.g. not initialized EEPROM by `clearAllBonds` before new bonding).
	  • Serial EEPROM access error. Additionally, the `_eeeError` flag is set in this case.

Output values	• 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. (`setCoordinatorMode` must be called first.) • Authorized Node must already be prebonded. • Authorized Node must be in the answerSystemPacket loop routine during Athorization.
Remarks	• See example E11-IQMESH-DFM-C [10] and IQRF OS User's guide – Bonding. • This function sends an authorization packet (broadcast), does not wait for delivery and provides no checking. Therefore it is necessary to wait before sending the packet testing the result of the authorization till the authorization packet itself reaches the destination. This period to wait can be evaluated from `RTHOPS` and `RTTSLOT`. See *Example*. • An occupied address can be unblocked by `removeBondedNode(address)`. • RF power and RF channel is not affected. The STD mode is used. • In case of prebonding by the Coordinator and keeping the `_prebondMode` bit set in Coordinator, the authorization packet is sent without routing which significantly speeds up the authorization. After the authorization, the `_prebondMode` bit can be cleared or left set as needed.
Side effects	• `PIN`, `DLEN`, `bufferRF` and `bufferINFO` are modified

See also	`bondNewNode, prebondNode, removeBondedNode, isBondedNode, setUserAddress, setCoordinatorMode, bondRequestAdvanced, answerSystemPacket, removeBondAddress`

Example	`uns16 mid;`
	`mid.high8 = MID_0;`
	`mid.low8 = MID_1;`
	
	`if (nodeAuthorization(mid, address))  // Was the address free?`
	`{ // Yes:`
	`  waitNewTick(); // Wait for end of nodeAuthorization packet routing`
	`  while (RTHOPS) // RTHOPS - rest of hops`
	`  {`
	`  waitDelay(RTTSLOT); // RTTSLOT - timeslot`
	`  RTHOPS--; // Do not answer until all hops are finished`
	`  }`
	`  ... // Send test packet to given address`
	
	`  if (ping successful)`
	`  { // Yes: successful authorization`
	`  ...`
	`  }`
	`  else`
	`  { // No: unsuccessful authorization`
	`  removeBondedNode(address);`
	`  }`
	`}`
	`else`
	`{ // No:`
	`  ... // Address is occupied`
	`}`
	`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(node)`**
Parameters	`uns8 node`: 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 Coordinator only. (`setCoordinatorMode` must be called first.)
Remarks	–
Side effects	–
See also	`bondNewNode, removeBondedNode, rebondNode, clearAllBonds, nodeAuthorization`

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(node)`**
Parameters	`uns8 node`: 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 Coordinator only. (`setCoordinatorMode` must be called first.)
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(node)`**
Parameters	`uns8 node`: 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 Coordinator only. (`setCoordinatorMode` must be called first.) • 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 Coordinator only. • `clearAllBonds` must be used to initialize serial EEPROM before creating the IQMESH network (before the first bonding).
Remarks	• See example E11-IQMESH-DFM-C [10]. • After subsequent bondNewNode(0) the Coordinator will start to assign Node numbers from 0.
Side effects	`bufferINFO` modified
See also	`removeBondedNode`
Example	`clearAllBonds(); // Exclude all currently bonded nodes from the network`

Bonding – Node and Coordinator

prebondNode

Function	Look for bond requesting devices and prebond a new Node based on its request via RF. Assign the universal address 0xFE and Network ID and send both to the new Node via RF and get back Module ID (MID) of this new Node.
Purpose	Prebond a new Node to the network. This is the first phase of the remote bonding process and possibly operates even without direct action of the Coordinator. Then it is needed to deliver the obtained MID to the Coordinator (the second phase, not necessary for prebonding by the Coordinator) by the application program. To complete remote bonding, the Coordinator must authorize the new Node (the third phase of the remote bonding process). Moreover, a 2 B user data is exchanged between prebonded device and the device providing prebonding.
Syntax	`bit preBondNode()`
Parameters	–
Return value	• 1 – prebonding successful. `bufferINFO` contains MID of prebonded Node • 0 – prebonding unsuccessful. `bufferINFO` unchanged

Input values	• Flag `_prebondMode`:
	  • 0: prebonding device works as a Node (default).
	  • 1: prebonding device works as the Coordinator.
	• uns16 `DataInPrebondNode`: User data to be delivered to the prebonded Node (to `DataOutBondRequestAdvanced`)

Output values	• `bufferINFO[0 to 3]`: MID of the prebondedNode (if successfully prebonded), LSB in `bufferINFO[0]`. • uns16 `DataOutPrebondNode`: User data received from the prebonded Node (from `DataInBondRequestAdvanced`)
Preconditions	• For IQMESH only (`setNodeMode` or `setCoordinatorMode` must be called first). • The Node providing prebonding must already be bonded in given network (to be able to provide the Network ID) and must be accessible from the Coordinator (either directly or via other Nodes). This is no use in case of prebonding by the Coordinator. • While the new Node is requesting to be prebonded (by function `bondRequestAdvanced`), the `prebondNode` function must be called in RF RX loop. See the Example below.
Remarks	• See example E11-IQMESH-DFM-N [10] and IQRF OS User's guide, chapter Bonding. • It is recommended to activate this function in application SW just in periods when it is actually needed. See the `prebondingEnabled` flag in the Example below. Permanent activation can ask for troubles with unwanted prebonding. If more Nodes have activated prebonding, it should be found out which Node has prebonded the new Node and ensure delivering of received MID to the Coordinator. • RF power and RF channel is not affected. The STD mode is used. • The function responds to bond request only if the following internal condition is valid: `if (((counter ^ address) & bondingMask) == 0)` `counter` Request for bonding contains a counter cleared after reset and incremented after every `bondRequestAdvanced()` call. • `address` Address of the device providing prebonding (evaluating the condition above). • `bondingMask` User-accessible register (with default value 0, i.e. the condition is true independently on the `counter` as well as the `address`). See IQRF User's guide, chapter Bonding for more information about the `bondingMask` usage.
Side effects	The following values are modified and not restored: • `PIN`, `DLEN`, `bufferRF` and `bufferINFO` are modified • IQMESH mode must be restored by `setNodeMode` or `setCoordinatorMode` after `prebondNode` • A/D converter control registers are modified
See also	`bondRequestAdvanced, setNodeMode, setCoordinatorMode,nodeAuthorization, answerSystemPacket`

Example

```
if (RFRXpacket())
{
  ...
}
else
{
  DataInPrebondNode = user_in_value;  // uns16, to be delivered to prebonded
                 // Node (to register DataOutBondRequestAdvanced)
                          // Try prebonding if enabled by user
  if (prebondingEnabled && prebondNode())
  {
    setNodeMode();        // Restore
    ...                   // MID of prebonded Node is stored in bufferINFO
    uns16 user_out_value = DataOutPrebondNode;  // A value received from
                 // prebonded Node (from register DataInBondRequestAdvanced)
  }
  ...
}
```

RFPGM – wireless upload

enableRFPGM

Function	Request to configure OS for switching to RFPGM mode after TR module reset
Purpose	Enable switching to RFPGM mode after reset
Syntax	`void` **`enableRFPGM()`**
Parameters	–
Return value	–
Output values	IQRF OS is reconfigured. This function overrides the setting by *TR Configuration* performed in IQRF IDE.
Preconditions	
Remarks	This function must be executed first to modify OS and just the following reset will switch to RFPGM.
Side effects	–
See also	`disableRFPGM`, `runRFPGM`, `setupRFPGM`

Example 1	`void APPLICATION()`
	`{`
	`enableRFPGM();`
	`  ...`

Example 2	See `disableRFPGM`

disableRFPGM

Function	Request to configure OS for not switching to RFPGM mode after TR module reset
Purpose	Disable switching to RFPGM mode after reset
Syntax	`void` **`disableRFPGM()`**
Parameters	–
Return value	–
Output values	IQRF OS is reconfigured. This function overrides the setting by *TR Configuration* performed in IQRF IDE.
Preconditions	–
Remarks	This function must be executed first to modify OS and just the following reset will not switch to RFPGM.
Side effects	–
See also	`enableRFPGM`, `setupRFPGM`

Example 1	`enableRFPGM();  // During development`
	`// disableRFPGM();`

Example 2	`// enableRFPGM();` `disableRFPGM();  // For final application`

runRFPGM

Function	Switch to RFPGM mode
Purpose	One-shot immediate switching to RFPGM mode
Syntax	`void` **`runRFPGM()`**
Parameters	–
Return value	–
Output values	RFPGM mode initiated
Preconditions	• For non-networking modes and RF bit rate 19.836 kb/s only. • All user peripherals (UART, Timer6,…) used must have their interrupt enable flags (`TXIE`, `TMR6IE`,…) disabled first. • LP mode must be activated in IQRF IDE when uploaded TR modules use low power RFPGM mode.

Remarks	• RF programming uses RF band and RF channel according to TR module configuration.
	• RFPGM mode can be refused:
	  • By low level on dedicated pin (if enabled). See `setupRFPGM` Parameters.
	  • By the *End RFPGM* button in IQRF IDE (unconditionally)
	  • ~1 minute after entering RFPGM mode (if enabled)
	• After RFPGM finishing the application is always reset (regardless to RFPGM upload result).
	• After unsuccessful RFPGM upload the TR stays in RFPGM mode, see IQRF OS User's guide, Appendix 3 (RFPGM).

Side effects	–
See also	`enableRFPGM, setupRFPGM`

Example 1

```
void APPLICATION()
...
if (jumperSet)
{
  runRFPGM();                 // Enter RFPGM mode on special request
}
```

Example 2

```
                              // All user peripheral interrupts must be
                              // disabled here (if used)
RCIE = 0;                     // E.g. UART RX interrupt disable
TMR6IE = 0;                   // E.g. Timer 6 interrupt disable

runRFPGM();                   // Run on channel(s)according to TR configuration
```

setupRFPGM

Function	Setup RFPGM parameters
Purpose	Configure RFPGM behavior
Syntax	`void` **`setupRFPGM(x)`**
Parameters	`uns8 x:` Factory default: 0x83

bit	7	6	5	4	3	2	1	0
	RFPGM termination by MCU pin(s)	RFPGM termination after ~1 min	0	RFPGM enable	0	LP RFPGM	Single / dual channel	

Bit 0,1: RFPGM single / dual channel mode
- 00 Receiving on single channel
- 01 Reserved
- 10 Reserved
- 11 Receiving on dual channel (default, can be changed by *TR Configuration* in IQRF IDE)

Bit 2: LP RFPGM
- 0 Uploaded TRs uses STD RX mode (default).
- 1 Uploaded TRs uses power saving LP RX mode.

Bit 4: RFPGM invoking by reset. H – enabled, L – disabled (default). This bit operates like `enableRFPGM` / `disableRFPGM` functions.

Bit 6: RFPGM termination automatically ~1 minute after entering RFPGM mode. H – enabled, L – disabled (default)

Bit 7: RFPGM termination by MCU pins RA5 or RB4. H – enabled (default), L – disabled. If enabled, the termination is invoked by log. 0 for at least ~0.25 s for single channel or ~0.5 s for dual channel on one of the dedicated pin(s):
- C5 for TR modules with shared RB4 and RA5 MCU pins on the C5 SIM pad, e.g. TR-72D
- Q2 or Q3 for TR-75D
- Q9 or Q12 for TR-76D

Bits 2, 3, and 5: Must be kept cleared

Return value	–
Output values	OS is modified and setup values are applicable anytime later.
Preconditions	If RFPGM termination by MCU pins is enabled, both RB4 and RA5 pins must have pull-up resistors. RB4 has an internal pull-up, default enabled by OS after boot. RA5 must have an external pull-up for TR modules with not shared RB4 and RA5 MCU pins on the C5 SIM pad.
Remarks	• RFPGM invoking by runRFPGM() is unconditional, independent on parameter x • RFPGM termination by IQRF IDE is unconditional, independent on parameter `x` • This function overrides the setting done by *TR Configuration* in IQRF IDE.
Side effects	–
See also	`runRFPGM, enableRFPGM`

Example 1

```
void APPLICATION()
setupRFPGM(0x13);     // RFPGM entered:   after reset or runRFPGM
                      // RFPGM abandoned: by End RFPGM button only
                      // Dual channel
```

Example 2

```
setupRFPGM(0x90);     // RFPGM entered:   after reset or runRFPGM
                      // RFPGM abandoned: by dedicated pin or End RFPGM button
                      // only
                      // Single channel
```

Example 3

```
setupRFPGM(0xD3);     // RFPGM entered:   after reset or runRFPGM
                      // RFPGM abandoned: by dedicated pin or End RFPGM button
                      // or automatically ~1 min after reset
                      // Dual channel
```

Documentation and information

1 **IQRF OS User's guide**
2 **RAM map** and **EEPROM map**, IQRF OS User's guide, Appendix 1 [1]
3 **IQRF home page**
4 **IQMESH specification**
5 **SPI specification**
6 **IQRF support**
7 **TR-72D datasheet**
8 **PIC16LF1938 datasheet**
9 **IQRF IDE**
10 **Examples** (included in the StartUp Package)

If you need a help or more information please contact IQRF support [6]. A lot of information is also available in the IQRF OS User's guide [1] and on the IQRF web site [3].

Document revision

- 150811 First release.

Index

Sales and Service

Corporate office

MICRORISC s.r.o., Prumyslova 1275, 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 : 2009 certified

Trademarks

Legal

Website	**www.iqrf.org**
E-mail	**sales@iqrf.org**
On-line support	**support@iqrf.org**

Smarter wireless. Simply.