IQRF OS

Operating System

Version 4.02D for TR-7xD

Reference Guide

1 Quick reference

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 IQRF OS User's guide [1], chapter *RAM map*. 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` or `startDelay` run in OS background. Thus, they are not blocking. The program execution continues immediately further and the user can check the result later on.

2 Table of OS functions

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

Control	
`wasRFICrestartedRFIC()`	Check RF IC functionality and possibly perform RF IC reset
`iqrfSleep()`	Set the TR module in power saving mode (Sleep)
`iqrfDeepSleep()`	Set the TR module in extremely power saving mode (Deep sleep)
`setRFsleep()`	Set the RF IC in power saving mode (Sleep)
`setRFready()`	Set the RF IC in ready mode (wake-up from Sleep)
`debug()`	Enter the debug mode
`uns8 getSupplyVoltage()`	Get voltage level for battery check
`int8 getTemperature()`	Temperature measurement
Active (blocking) waiting	
`waitMS(ms)`	Active waiting (time in ms)
`waitDelay(ticks)`	Active waiting (time in ticks)
`waitNewTick()`	Wait for a new tick
Timing on background	
`startCapture()`	Resets counter of ticks
`captureTicks()`	Get number of ticks counted
`startDelay(ticks)`	Start waiting (time in ticks)
`startLongDelay(ticks)`	Start long waiting (time in ticks)
`bit isDelay()`	Still waiting
LED indication	
`setOnPulsingLED(ticks)`	LEDR and LEDG On times setting (for blinking)
`setOffPulsingLED(ticks)`	LEDR and LEDG Off times setting (for blinking)
`pulsingLEDR()`	Red LED activation (blinking on background)
`pulseLEDR()`	Single red LED pulse (one flash on background)
`stopLEDR()`	Red LED off, blinking stopped
`pulsingLEDG()`	Green LED activation (blinking on background)
`pulseLEDG()`	Single green LED pulse (one flash on background)
`stopLEDG()`	Green LED off, blinking stopped
MCU EEPROM	
`uns8 eeReadByte(address)`	Read one byte
`eeReadData(address, length)`	Read a block
`eeWriteByte(address, data)`	Write one byte
`eeWriteData(address, length)`	Write a block
Serial EEPROM	
`bit eeeReadData(address)`	Read a 16 B block from serial EEPROM to bufferINFO
`bit eeeWriteData(address)`	Write a 16 B block from bufferINFO to EEPROM
RAM	
`uns8 readFromRAM(address)`	Read one byte
`void setINDF0(value)`	Indirect write via virtual INDF0 register
`void setINDF1(value)`	Indirect write via virtual INDF1 register

Buffers	
copyBufferINFO2COM()	Copy bufferINFO to bufferCOM
copyBufferINFO2RF()	Copy bufferINFO to bufferRF
copyBufferRF2COM()	Copy bufferRF to bufferCOM
copyBufferRF2INFO()	Copy bufferRF to bufferINFO
copyBufferCOM2RF()	Copy bufferCOM to bufferRF
copyBufferCOM2INFO()	Copy bufferCOM to bufferINFO
bit compareBufferINFO2RF(length)	Comparison of bufferINFO and bufferRF
void swapBufferINFO()	Swap bufferINFO and bufferAUX
clearBufferINFO()	bufferINFO clearing
clearBufferRF()	bufferRF clearing
Data blocks	
copyMemoryBlock (uns16 from, uns16 to, uns8 length)	Copy any data block to any position
moduleInfo()	Get info about transceiver module and OS
SPI	
enableSPI()	SPI communication line activation
disableSPI()	SPI communication line deactivation
startSPI(length)	SPI packet transmission
stopSPI()	SPI stopping
restartSPI()	SPI continuing
bit getStatusSPI()	SPI status, update SPI flags
RF	
setRFpower(level)	RF TX power setting (8 levels)
setRFspeed(speed)	Select RF bit rate – not yet implemented
setRFchannel(channel)	Select RF channel
setRFmode(mode)	Select RF power management mode
checkRF(level)	Detect incoming RF signal
getRSSI()	Get RSSI value of incoming RF signal
RFTXpacket()	Send a packet from bufferRF via RF
bit RFRXpacket()	Receive a packet via RF to bufferRF
Networking	
setCoordinatorMode()	Device is the Coordinator
setNodeMode()	Device is a Node
setNonetMode()	Networking disabled
setNetworkFilteringOn()	Packets accepted from current network only
setNetworkFilteringOff()	Packets accepted from both networks
uns8 getNetworkParams()	Get information about the network
void sendFRC(cmd)	Request for Fast Response Command
void responseFRC()	Answer to Fast Response Command
bit amIRecipientfOfFRC()	Evaluate whether the FRC conmmand is intended for given Node

Routing	
setRoutingOn()	Outgoing packets routed via other devices on background
setRoutingOff()	No routing for outgoing packets
uns8 discovery(MaxNodeNumber)	Discover Nodes for routing
answerSystemPacket()	Enable response to Coordinator for Discovery and nodeAuthorization
bit isDiscoveredNode(N)	Check for being discovered
bit wasRouted()	Indicate incoming packet routing
optimizeHops(method)	Optimize number of hops for given Node
Bonding - Node	
bit bondRequestAdvanced()	Request for bonding (local or remote)
bit amIBonded()	Is the Node bonded?
removeBondAddress()	Changing Node address to universal address (0xFE)
removeBond()	Unbonding
prebondNodeAtNode()	Preparing Node for remote bonding via another Node
Bonding - Coordinator	
bit bondNewNode(address)	Local bonding a Node
bit prebondNodeAtCoordinator(address)	Preparing Node for remote bonding via Coordinator
nodeAuthorization()	Remote bonding of prebonded Node
bit isBondedNode(node)	Is the Node bonded?
removeBondedNode(node)	Unbonding a Node
bit rebondNode(node)	Rebonding a Node
clearAllBonds()	Clearing of all bonds
Encryption	
void setAccessPassword()	Set Access password
void setUserKey()	Set the key for user encryption and decryption
void encryptBufferRF(W)	Encrypt bufferRF
void decryptBufferRF(W)	Decrypt bufferRF
RFPGM	
enableRFPGM()	Set to switch to RFPGM mode after reset
disableRFPGM()	Set not to switch to RFPGM mode after reset
runRFPGM()	Switch to RFPGM mode
setupRFPGM(x)	Setup RFPGM parameters

3 Table of macros

Constants	
Control	
reset()	Restart MCU, IQRF OS and application SW
setBORon()	Enable MCU Brown-out reset
setBORoff()	Disable MCU Brown-out reset
setWDTon()	Enable Watchdog
setWDToff()	Disable Watchdog
setWDTon_xxxx()	Enable Watchdog with wake-up after specifid time
sleepWOC()	TR Sleep with wake-up on change at dedicated TR pin enabled
setIOCBN()	Set the MCU flag IOCBN4
clearIOCBN()	Clear the MCU flag IOCBN4
breakpoint(wValue)	Call debug with specified value in w register
bit buttonPressed()	Read level at dedicated pin
Serial EEPROM and temperature sensor	
eEEPROM_TempSensorOn()	Enable serial EEPROM and temperature sensor
eEEPROM_TempSensorOff()	Disable serial EEPROM and temperature sensor
RAM	
writeToRAM(address, data)	Write one byte
uns8 setFSR0(buffer)	Set control register FSR0 to access specified OS buffer
uns8 setFSR1(buffer)	Set control register FSR1 to access specified OS buffer
uns8 setFSR01(buffer0, buffer1)	Set control registers FSR0 and FSR1 to access specified OS buffers
Data blocks	
appInfo()	Copy info about application from EEPROM to bufferINFO
Networking	
setFRCresponseTime(ms)	Specify the time needed to complete FRC responses in Nodes
uns8 getFRCresponseTime()	Get current value of FRC response time
Compatibility	

4 OS functions

4.1 Control

4.1.1 wasRFICrestarted

Function	Restart RF IC if it is required after internal failure
Purpose	To check whether an RF IC failure (e.g. the oscillator malfunction) has been detected. If so, OS automatically performs RF IC reset and the user should restore non-default RF parameters then.
Syntax	`uns8 wasRFICrestarted()`
Parameters	–
Return value	• 1 RF IC failure detected and RF IC reset has been performed. • 0 No RF IC failure detected, no RF IC reset has been performed.
Output values	–
Preconditions	• To be checked after `RFTXpacket` and `RFRXpacket` • It is recommended to implement this check in main loop in application SW, especially when high robustness is required.
Remarks	• If RF IC restart has been performed, all RF parameters specified by the user (RF channel, TX power, possibly RF band and parameters set by the `setRFmode` and `checkRF`) which are different from OS default and parameters specified in TR configuration must be restored first. See Example. • If RF IC reset is performed, it takes about 100 ms.
Side effects	–
See also	`reset`

Example	`while (1)`
	`{`
	`  ...`
	`  RFTXpacket();`
	`  if (wasRFICrestarted())`
	`  {`
	`  setRFmode(0x50);`
	`  ...`
	`  }`

4.1.2 iqrfSleep

Function	Setting the TR module in power saving mode (Sleep)
Purpose	Easy and efficient power management. This function, puts the TR into the Sleep mode.
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 (RF IC is put into Sleep mode), 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. • No PIC pins must be left as digital inputs without defined input log. level values. See example E14-CONSUMPTION. • Global interrupt enable (`GIE`) must not be disabled before `iqrfSleep` call. • For wake-up on pin change the required sequence shoud be executed, see the Example 2 below. Macro `sleepWOC()` can be used for this. 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 can be caused by power off/on, watchdog timeout or on the C5 (for TR modules for SIM mounting, e.g. TR-72D) or Q12 (for SMT mounting, e.g.TR-76D) pin change. • Wake-up types can be identified via the `-TO` and `-PD` status flags (in the MCU `STATUS` register).
Side effects	–
See also	`setRFsleep`, `iqrfDeepSleep`, `sleepWOC`

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 enabled.`
	`// Instead of IOCBN.4=1; Bit IOCBN.4 cannot`
	`// be written directly due to OS restriction`
	`IOCBP.4 = 1;  // Positive edge enabled`
	`IOCIE = 1;  // Interrupt on change enabled`
	`GIE = 1;  // Global interrupt enabled`
	`SWDTEN = 0;  // Watchdog disabled`
	`iqrfSleep();  // Sleep`
	`GIE = 0;`
	`writeToRAM(&IOCBN, IOCBN & 0xEF);  // Negative edge disabled (Instead of IOCBN.4=0)`
	`IOCBP.4 = 0;  // Positive edge enabled`
	`GIE = 1;  // Global interrupt disabled`
	`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`

4.1.3 iqrfDeepSleep

Function	Setting the TR module in extremely power saving mode (Deep sleep). This function operates like the `iqrfSleep` but RF IC is put in the Shutdown (with no internal supply of RF circuitry) instead of the Sleep state.
Purpose	Power management in cases when extreme low power consumption is required and TR operation can be disabled for long periods. This function, puts TR including all RF IC functionality into the Deep sleep mode.
Syntax	`void` **`iqrfDeepSleep()`**
Parameters	–
Return value	–
Output values	After waking up, RF IC will be switched to RF Sleep mode and reset to default state like after power on.
Preconditions	• The user should suspend all resources used by the application before entering the Deep sleep mode to achieve minimal power consumption. • No PIC pins must be left as digital inputs without defined input log. level values. See example E14-CONSUMPTION. • Global interrupt enable (`GIE`) must not be disabled before `iqrfDeepSleep` call. • For wake-up on pin change the required sequence should be executed, see the Example 2 below. Wake-up on pin change is default disabled.
Remarks	• `IOCBF` flag is cleared automatically by OS. • Flags `IOCBN` and `IOCBP` are unchanged (not cleared) within `iqrfDeepSleep`. • Wake-up can be caused by power off/on, watchdog timeout or on the C5 (for TR modules for SIM mounting, e.g. TR-72D) or Q12 (for SMT mounting, e.g.TR-76D) pin change. • Wake-up types can be identified via the `-TO` and `-PD` status flags (in the MCU `STATUS` register). • If RF functionality is needed after waking up, the `setRFready` must be called and all RF parameters specified by the user (RF channel, TX power, possibly RF band and parameters set by the `setRFmode` and `checkRF`) which are different from OS default and parameters specified in TR configuration must be restored first. See Example 4.
Side effects	–
See also	`iqrfSleep`, `setRFsleep`

Example 1	`// Minimize consumption (depends on resources used by the user)`
	`  ...  // Disable all TR resources utilized by the user`
	`SWDTEN = 0;  // Disable watchdog`
	`iqrfDeepSleep();  // Put the module into Deep sleep mode`

Example 2	`  // Wake-up on pin change.`
	`GIE = 0;  // Disable all interrupts`
	`writeToRAM(&IOCBN, IOCBN | 0x10);  // Negative edge enabled.`
	`  // Instead of IOCBN.4=1;`
	`  // Bit IOCBN.4 cannot be written`
	`  // directly due to OS restriction.`
	`IOCBP.4 = 1;  // Positive edge enabled`
	`IOCIE = 1;  // Interrupt on change enabled`
	`GIE = 1;  // Global interrupt enabled`
	`SWDTEN = 0;  // Watchdog disabled`
	`iqrfDeepSleep();  // Sleep`
	`GIE = 0;`
	`writeToRAM(&IOCBN, IOCBN & 0xEF);  // Negative edge disabled (Instead of IOCBN.4=0)`
	`IOCBP.4 = 0;  // Positive edge disabled`
	`GIE = 1;  // Global interrupt enabled`
	`if (buttonPressed)  // If button is pressed`
	`  { ... }  //  ...`

Example 3	`iqrfDeepSleep();  // Deep sleep`
	`  ...  // Perform necessary operation (if no RF is needed)`
	`iqrfDeepSleep();  // and go to Deep sleep again as soon as possible`

Example 4	`iqrfDeepSleep();  // Deep sleep`
	`setRFready();  // After waking up: switch RF IC to Ready mode`
	`setRFchannel(40);  //  Restore all RF parameters to be specified by application`
	`setRFpower(5);`
	`setRFmode(0x51);`
	`checkRF(3);`
	`  ...  //  and continue`

4.1.4 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 [4] for power consumption saving.
Side effects	–
See also	`setRFready`, `iqrfSleep`, `iqrfDeepSleep`, `checkRF`, `RFTXpacket`, `RFRXpacket`
Example	`setRFsleep(); // Put the RF circuitry in Sleep mode`

4.1.5 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 [1], *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. However, MCU system clock always stays derived from internal RC oscillator.
Preconditions	–
Remarks	After the RF wake-up the RX chain can be set on faster which enables faster `checkRF`, `RFRXpacket` or `RFTXpacket`.
Side effects	–
See also	`setRFsleep`, `iqrfSleep`, `iqrfDeepSleep`, `checkRF`, `RFTXpacket`, `RFRXpacket`

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

4.1.6 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 [8] 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 [8] Help and Example E04-EEPROM [9].
Side effects	• `param1` to `param4`, `memoryOffsetTo`, `memoryOffsetFrom` and `memoryLimit` are not displayed • Watchdog is cleared while in Debug mode
See also	`breakpoint`

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`

4.1.7 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`

4.1.8 getTemperature

Function	Read temperature from on-board sensor
Purpose	Temperature measurement
Syntax	`int8` **`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 [7]. 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, after wake up from sleep or after `eEEPROM_TempSensorOn`.
Remarks	• Resolution 0.5 °C, accuracy: 0.5 °C • Takes about 3 ms. • See Example E08–TEMPERATURE [9].
Side effects	–
See also	–

Example 1

```
   // For positive temperatures only
int8   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 > (uns8)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() > (uns8)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();
```

4.2 Active (blocking) waiting

4.2.1 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 [6].
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.`

4.2.2 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`. • This function must not be combined with `startDelay` and `startLongDelay`.
Remarks	This is the active waiting (on OS foreground). No other operation runs on OS foreground during waiting.
Side effects	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 [6].
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.`

4.2.3 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;`

4.3 Timing on background

4.3.1 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 `bondRequestAdvanced`, `bondRequest`, `prebondNodeAtNode`, `prebondNodeAtCoordinator`, `nodeAuthorization`, `bondNewNode`, `sendFRC`, `responseFRC`, `discovery`, `answerSystemPacket`, `RFRXpacket` and `RFTXpacket`.
See also	`captureTicks`
Example	See `captureTicks`

4.3.2 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` or `startCapture`, whatever was the latest (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 [9].
Side effects	• Functionality is affected by `bondRequestAdvanced`, `bondRequest`, `prebondNodeAtNode`, `prebondNodeAtCoordinator`, `nodeAuthorization`, `bondNewNode`, `sendFRC`, `responseFRC`, `discovery`, `answerSystemPacket`, `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 [6].
See also	`startCapture`, `setRFready`

Example	`startCapture();  // Reset counter of ticks`
	`waitMS(200);  // Delay 200 ms`
	`captureTicks();  // param3 == 20, param4 == 20`
	`waitMS(150);  // Delay 150 ms`
	`captureTicks();  // param3 == 35, param4 == 15`
	`startCapture();  // Reset counter of ticks`
	`waitMS(100);  // Delay 100 ms`
	`captureTicks();  // param3 == 10, param4 == 10`

4.3.3 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 `bondRequestAdvanced`, `bondRequest`, `prebondNodeAtNode`, `prebondNodeAtCoordinator`, `nodeAuthorization`, `bondNewNode`, `sendFRC`, `responseFRC`, `discovery`, `answerSystemPacket`, `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 [6].
See also	`isDelay`, `startLongDelay`, `waitDelay`
Example	See `isDelay`

4.3.4 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 `bondRequestAdvanced`, `bondRequest`, `prebondNodeAtNode`, `prebondNodeAtCoordinator`, `nodeAuthorization`, `bondNewNode`, `sendFRC`, `responseFRC`, `discovery`, `answerSystemPacket`, `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 [6].
See also	`isDelay`, `startDelay`, `waitDelay`
Example	See `isDelay`

4.3.5 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 [9].
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`

4.4 LED indication

4.4.1 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`

4.4.2 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`

4.4.3 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`

4.4.4 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	`setOnPulsingLED`, `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`

4.4.5 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`

4.4.6 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`

4.4.7 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	`setOnPulsingLED`, `pulsingLEDG`, `stopLEDG`

Example	`setOnPulsingLED(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`

4.4.8 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`

4.5 MCU EEPROM

4.5.1 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 [9].
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`

4.5.2 eeReadData

Function	Read a block of specified length from specified location in EEPROM to `bufferINFO`
Purpose	Block access to EEPROM
Syntax	`bit` **`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 64)
Return value	• 0: only non-zero bytes has been read • 1: at least one zero byte has been read
Output values	• `bufferINFO[0` to `length - 1]` • `bufferINFO[0` to `length - 1]` is cleared when attempted to read from address 0xC0 or higher
Preconditions	Destination address in `bufferINFO` can be shifted by `memoryOffsetTo`. `memoryOffsetTo` is default disabled (cleared after reset as well as after every `eeReadData`).
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 [9].
Side effects	–
See also	`eeReadByte`, `eeWriteByte`, `eeWriteData`

Example 1	`eeReadData(0x0A, 16);  // copy 16 B 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]`

Example 3	`memoryOffsetTo = 20;`
	`eeReadData(0x0A, 16);  // copy 16 B from EEPROM from address 0x0A to bufferINFO`
	`//  bufferINFO[20] = EEPROM[0x0A]`
	`//  ...`
	`//  bufferINFO[35] = EEPROM[0x19]`
	`// memoryOffsetTo is automatically cleared here`

4.5.3 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 [9]. • 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.`

4.5.4 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 64)

Return value	–
Output values	–
Preconditions	Initial address in `bufferINFO` can be shifted by `memoryOffsetFrom`. `memoryOffsetFrom` is default disabled (cleared after reset as well as after every `eeWriteData`).
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 [9].
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 16 B 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.`

Example 3	`memoryOffsetFrom = 20;`
	`eeWriteData(0x0A,16);  // copy 16 B from bufferINFO to EEPROM to address 0x0A`
	`//  EEPROM[0x0A] = bufferINFO[20]`
	`//  ...`
	`//  EEPROM[0x19] = bufferINFO[35]`
	`// memoryOffsetFrom is automatically cleared here`

4.6 Serial EEPROM

4.6.1 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. Subsequent clearing of this flag is up to the user.
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 16 B is read. • To respect accessible range, the following rule must be observed: `address + memoryLimit < 0x8000`. See Example 2 and 3.
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 16 B`
	`// When memoryLimit is kept cleared from previous operations`
	`eeeReadData(0x3C);`
	`// copy 16 B from serial EEPROM from address 0x3C to bufferINFO`
	`// bufferINFO[0] = serial EEPROM[0x3C]`
	`//  ...`
	`// bufferINFO[15] = serial EEPROM[0x4B]`

Example 2	`memoryLimit = 40;  // To read 40 B`
	`eeeReadData(0x3C);  // copy 40 B 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`
	`  }`

4.6.2 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 (`0` to `0x3FFF`)
Return value	• 1 Write successful • 0 Write unsuccessful (e.g. due to damaged or not populated memory device). Additionally, the `_eeeError flag` is set. Subsequent clearing of this flag is up to the user.
Output values	Memory range `0` to `0x3FFF` 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 16 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`, …, `0x3FC0` – `0x3FFF`. See Example 3, 4 and 5.
Remarks	`memoryLimit` is automatically cleared after every `eeeWriteData` call.
Side effects	–
See also	`eeeReadData`

Example 1	`// To write 16 B`
	`// When memoryLimit is kept cleared from previous operations`
	`eeeWriteData(0x40);  // copy 16 B from bufferINFO to serial EEPROM`
	`// from address 0x40`
	`// EEPROM[0x40] = bufferINFO[0]`
	`//  ...`
	`// EEPROM[0x4F] = bufferINFO[15]`

Example 2	`// To write 64 B`
	`memoryLimit = 64;`
	`eeeWriteData(0x40);  // copy 64 B from bufferINFO to serial EEPROM`
	`// from address 0x40`
	`// EEPROM[0x40] = bufferINFO[0]`
	`//  ...`
	`// EEPROM[0x7F] = bufferINFO[63]`

Example 3	`memoryLimit = 6;  // To write 6 B`
	`eeeWriteData(0x40);  // copy 6 B from bufferINFO to serial EEPROM from`
	`// address 0x40`
	`// EEPROM[0x40] = bufferINFO[0]`
	`//  ...`
	`// EEPROM[0x45] = bufferINFO[5]`
	`// memoryLimit is automatically cleared here`

Example 4	`// To write 64 B`
	`memoryLimit = 64;`
	`eeeWriteData(0x41);  // Illegal access, boundary 0x80 crossed`

Example 5	`memoryLimit = 6;  // to write 6 B`
	`eeeWriteData(0x2C7E); // Illegal access, boundary 0x2C80 crossed`

Example 6	`eeeWriteData(0x4000); // Illegal access, attempt to write to area dedicated to OS`

Example 7	`memoryLimit = 1;  // To write 1 B`
	`bufferINFO[0] = 'A'`
	`if (!eeeWriteData(0x0A))`
	`...  // Error handling`

4.7 RAM

4.7.1 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 [9].
Side effects	`FSR0` register is modified.
See also	`writeToRAM`, `copyMemoryBlock`, `getINDF0`, `getINDF1`

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

4.7.2 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 [9]. • Another possibility (but more code consuming) is using the `writeToRAM` function.
Side effects	–
See also	`setINDF1`, `getINDF0`, `getINDF1`, `writeToRAM`, `copyMemoryBlock`

Example	`// Block memory copying from bufferRF to bufferINFO`
	`FSR0 = bufferINFO + 5;  // To`
	`for(i = 0; i < 8; i++)`
	`{`
	`  x = bufferRF[i];  // From`
	`  setINDF0(x);`
	`  FSR0++;`
	`}  // Result is the same as`
	`// copyMemoryBlock(bufferRF, bufferINFO + 5, 8)`

4.7.3 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 [9]. • Another possibility (but more code consuming) is using the `writeToRAM` function.
Side effects	–
See also	`setINDF0`, `getINDF0`, `getINDF1`, `writeToRAM`, `copyMemoryBlock`
Example	Similar as for `setINDF0`.

4.8 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).

4.8.1 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 [9].
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).`

4.8.2 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 [9].
Side effects	–
See also	`clearBufferINFO`, `copyBufferINFO2COM`, `copyBufferRF2COM`, `copyBufferRF2INFO`, `copyBufferCOM2RF`, `copyBufferCOM2INFO`, `compareBufferINFO2RF`, `copyMemoryBlock`
Example	`copyBufferINFO2RF();`

4.8.3 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 [9].
Side effects	–
See also	`clearBufferINFO`, `copyBufferINFO2RF`, `copyBufferINFO2COM`, `copyBufferRF2INFO`, `copyBufferCOM2RF`, `copyBufferCOM2INFO`, `compareBufferINFO2RF`, `copyMemoryBlock`
Example	`copyBufferRF2COM();`

4.8.4 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 [9].
Side effects	–
See also	`clearBufferINFO`, `copyBufferINFO2COM`, `copyBufferINFO2RF`, `copyBufferRF2COM`, `copyBufferCOM2RF`, `copyBufferCOM2INFO`, `compareBufferINFO2RF`, `copyMemoryBlock`
Example	`copyBufferRF2INFO();`

4.8.5 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 [9].
Side effects	–
See also	`clearBufferINFO`, `copyBufferINFO2COM`, `copyBufferINFO2RF`, `copyBufferRF2COM`, `copyBufferRF2INFO`, `copyBufferCOM2INFO`, `compareBufferINFO2RF`, `copyMemoryBlock`
Example	`copyBufferCOM2RF();`

4.8.6 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 [9].
Side effects	–
See also	`clearBufferINFO`, `copyBufferINFO2COM`, `copyBufferINFO2RF`, `copyBufferRF2COM`, `copyBufferRF2INFO`, `copyBufferCOM2RF`, `copyMemoryBlock`
Example	`copyBufferCOM2INFO();`

4.8.7 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	Offset `memoryOffsetFrom` is applied to shift initial address in `bufferINFO` and offset `memoryOffsetTo` is applied to shift initial address in `bufferRF`.
Remarks	• If `memoryOffsetFrom = 0`, `memoryOffsetTo = 0`, complete 64 B is compared. • See Example E06 - RAM [9].
Side effects	–
See also	`clearBufferINFO`, `copyBufferINFO2RF`, `copyBufferRF2INFO`, `swapBufferINFO`
Example	`if (!compareBufferINFO2RF(32))  // Compare 32 B` `  then Error = 1;  // Error if mismatch`

4.8.8 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 [9].
Preconditions	`memoryLimit` is applied to to swap less than 64 B. If `memoryLimit = 0` , complete 64 B is swapped.
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`

4.8.9 clearBufferINFO

Function	Clear `bufferINFO`
Purpose	`bufferINFO` clearing (filling with zeros)
Syntax	`void` **`clearBufferINFO()`**
Parameters	–
Return value	–
Output values	• If `memoryLimit == 0`, complete `bufferINFO` (64 B) is cleared. • If `memoryLimit <> 0`, just the first `memoryLimit` bytes of `bufferINFO` is cleared. See Example E06 - RAM [9].
Preconditions	Number of bytes to be cleared can be specified by `memoryLimit` (0 to 64).
Remarks	`memoryLimit` is automatically cleared after every `clearBufferINFO` call.
Side effects	–
See also	`copyBufferINFO2COM`, `copyBufferINFO2RF`, `copyBufferRF2INFO`, `copyBufferCOM2INFO`, `compareBufferINFO2RF`, `copyMemoryBlock`, `swapBufferINFO`
Example 1	`clearBufferINFO();`
Example 2	`memoryLimit = 32;` `clearBufferINFO();  // Only the first half of bufferINFO is cleared`

4.8.10 clearBufferRF

Function	Clear `bufferRF`
Purpose	`bufferRF` clearing (filling with zeros)
Syntax	`void` **`clearBufferRF()`**
Parameters	–
Return value	–
Output values	• If `memoryLimit == 0`, complete `bufferRF` (64 B) is cleared. • If `memoryLimit <> 0`, just the first `memoryLimit` bytes of `bufferRF` is cleared. See Example E06 - RAM [9].
Preconditions	Number of bytes to be cleared can be specified by `memoryLimit` (0 to 64).
Remarks	`memoryLimit` is automatically cleared after every `clearBufferRF` call.
Side effects	–
See also	`copyBufferRF2COM`, `copyBufferRF2INFO`, `copyBufferCOM2RF`, `copyBufferINFO2RF`, `compareBufferINFO2RF`, `copyMemoryBlock`
Example 1	`clearBufferRF();`
Example 2	`memoryLimit = 32;` `clearBufferRF();  // Only the first half of bufferRF is cleared`

4.9 Data blocks

4.9.1 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 [9].
Side effects	`FSR0` and `FSR1` registers are modified.
See also	`writeToRAM`, `readFromRAM`, `setINDF0`, `getINDF0`, `setINDF1`, `getINDF1`
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`

4.9.2 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 / Module ID				OS version	TR type	OS build	

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

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
- 4: (DC)TR-78D
- 8: (DC)TR-54D
- 9: (DC)TR-55D
- 10: (DC)TR-56D
- 11: (DC)TR-76D
- 12: (DC)TR-77D
- 13: (DC)TR-75D

FCC
- 0: FCC not certified
- 1: FCC certified

MCU type
- 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  42  24  91  08
```

MID = `0100101C`, IQRF OS v4.02D, TR-72D, PIC16LF1938, FCC not certified, OS build `0x0891`.

```
bufferINFO[0-7] =  1C  10  00  81  42  BC  91  08
```

MID = `8100101C`, IQRF OS v4.02D, TR-76D, PIC16LF1938, FCC certified, OS build `0x0891`.

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 [3].
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
```

4.10 SPI

4.10.1 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 [3] and Example E07-SPI [9].
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`

4.10.2 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 [3] and Example E07-SPI [9].
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`

4.10.3 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 [3] and Example E07-SPI [9].
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`

4.10.4 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 [3] and Example E07-SPI [9].
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`
	`}`

4.10.5 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`

4.10.6 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 [3] and Example E07-SPI [9].
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
```

4.11 RF

4.11.1 setRFpower

Function	Set RF output power
Purpose	Change RF range
Syntax	`void` **`setRFpower(level)`**
Parameters	`uns8 level`: 0 (min.) to 7 (max. – default) See datasheet of TR module [4].
Return value	–
Output values	Available read only in the `RFpower` register
Preconditions	–
Remarks	–
Side effects	–
See also	`RFTXpacket`
Example	`setRFpower(7);  // Max. RF output power`

4.11.2 setRFspeed

Function	Select RF bit rate. **Not implemented** yet. Do not use this function. Bit rate 19.8 kb/s is fixed.
Purpose	Select RF bit rate
Syntax	`void` **`setRFspeed(speed)`**
Parameters	`uns8 speed`:
Return value	–
Output values	
Preconditions	
Remarks	
Side effects	
See also	`setRFchannel`
Example	

4.11.3 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 [1], *Appendix 2, Channel map* • Default: 868 MHz band: 52 916 MHz band: 104 433 MHz band: 8 The default channel can be changed by TR Configuration in IQRF IDE [8].
Return value	–
Output values	Available read only in the `RFchannel` register
Preconditions	• To avoid interferences between adjacent RF channels, the selection should respect the following rules (typical, in most cases): • STD mode: There are no interferences even between very adjacent channels. • LP or XLP modes: 10 channels spacing is required at worst case. Channels not interfering each other can be used in two overlapping IQRF networks transmitting at the same time. Interferences between two IQRF transceivers in LP or XLP modes significantly decrease with the distance between those transceivers. Examples for interference between two IQRF transceivers: • Channels 50 and 51 typically do not interfere in STD mode at any distance. • Channels 50 and 60 or higher typically do not interfere in all modes at any distance. • Channels 50 and 51 may typically interfere in LP or XLP at 1 m distance. • Channels 50 and 51 typically do not interfere in LP or XLP at 20 m distance. But in all cases it is recommended to observe spacing as high as possible.
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`

4.11.4 setRFmode

Function	Set modes for RF operation
Purpose	Specify RF RX and RF TX power modes and conditions for RX termination.
Syntax	`void setRFmode(mode)`
Parameters	`uns8 mode`: `PWTTNFRR` in binary `P` **Preamble length in STD TX** 0 Standard preamble length (4 ms) 1 Prolonged preamble length (8 ms). Suitable e.g. when working at two different RF channels. `W` **Wait packet end** 0 Terminate `RFRXpacket` unconditionally when `toutRF` expired. 1 Do not terminate `RFRXpacket` (ignore `toutRF` expiry) if the packet is currently receiving. `TT` **TX mode** 00 STD TX mode (standard ~4 ms or prolonged ~8 ms preamble) 01 LP TX mode (prolonged preamble ~50 ms) 10 XLP TX mode (prolonged preamble ~900 ms) 11 Reserved, do not use `N` Reserved for future use `F` Enable immediate RX termination (before `toutRF` timeout) when low level on the C5 (for TRs for SIM mounting, e.g. TR-72D) or Q12 (for TRs for SMT mounting, e.g.TR-76D) pin occurred. For low LP RX and XLP RX modes only. `RR` **RX mode** 00 STD RX mode (Standard RX). Use STD TX mode at the counterpart. 01 LP RX mode (Low power RX). Use LP TX or XLP TX mode at the counterpart. 10 XLP RX mode (Extra low power RX). Use XLP TX mode at the counterpart. 11 Reserved, do not use.
Return value	–
Output values	Available read only in the `RFmodeByte` register.
Preconditions	Default value is `mode = 0`.
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 to 4 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 4.
See also	`checkRF`

Example 1	`// RX: STD, TX: for STD RX (standard, preamble 4 ms)`
	`setRFmode(0b00000000);  // Using numeral value`
	`setRFmode(_RX_STD | _TX_STD)  // The same using predefined constants for clarity`

Example 2	`// RX: LP, TX: for LP RX (prolonged preamble ~50 ms)`
	`setRFmode(0b00010001);  // Using numeral value`
	`setRFmode(_RX_LP | _TX_LP)  // The same using predefined constants for clarity`

Example 3	`// RX: STD, TX: for STD RX (standard, preamble 8 ms)`
	`setRFmode(0b10000000);  // Using numeral value`
	`setRFmode(_RX_STD | _TX_STD | _STDL)  // The same using predefined constants for`
	`// clarity`

Example 4	`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 5

```
// RFRXpacket terminated after low level on C5/Q12 is detected and
// current cycle is finished.

toutRF = 100;              // [in cycles], 1 cycle = ~790 ms

while (1)
{
  setRFmode(_RX_XLP | _RLPMAT);  // RX_XLP + LP/XLP RX termination
  if (RFRXpacket())
  {
    ...
  }

  …                        // Goes here after every 79 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 6

```
// 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 = ~790 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 79 s (toutRF=100) or
                         //   immediately if low level appears on the C5/Q12 pin

  if (buttonPressed)
  {
    ...
    setRFmode(_RX_STD);                 // Set required TX preamble
    RFTXpacket();
  }
}
```

4.11.5 checkRF

Function	• Check currently incoming RF signal strength and preamble quality • Specify level of RSSI for subsequent receipts in LP and XLP modes
Purpose	• Incoming RF signal strength and quality detection • Set filter level for incoming LP and XLP packets
Syntax	`bit` **`checkRF(level)`**
Parameters	`uns8 level = RSSI_FILTER (0 to 64)` Values > 64 are not intended for signal filtration but for special purposes. See `getRSSI` Preconditions for example.
Input values	`bit _checkRFcfg_PQT` Preamble quality check enable in STD RX • 0: RF carrier strength is checked (Default) • 1: RF carrier strength and preamble quality are checked This bit applies in STD RX mode only. In LP or XLP RX modes the preamble quality is always checked.
Return value	• 1: If `_checkRFcfg_PQT = 0` in STD RX mode: Signal with specified level or higher detected (RSSI >= RSSI_FILTER) If `_checkRFcfg_PQT = 1` or in LP or XLP RX mode: Signal with specified level or higher detected (`RSSI >= RSSI_FILTER`) and correct format of packet preamble is detected • 0: Otherwise
Output values	• Filter for all subsequent incoming LP and XLP packets is set to specified method and level • 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 repeatedly in RX loop.
Remarks	• Higher filtration brings higher immunity against noise and interefrences but allows lower range. See TR datasheet [4], table *Relative RF range vs.* *`checkRF(level)`*. • For environment without a significant noise `checkRF(0)` is recommended. • Checking takes about 1 ms (when `_checkRFcfg_PQT=0`) or 2.8 ms (when `_checkRFcfgPQT=1`). • If `_checkRFcfgPQT=1` and a packet transmitted in LP or XLP TX mode should be received in STD RX mode, `toutRF ≥ 5` or `100`, respectively, must be set. • Unlike `getRSSI` and `RFRXpacket`, `checkRF` does not update the `lastRSSI` register. • For reading out the RSSI value the `getRSSI` function is intended. See `getRSSI` Example.
Side effects	–
See also	`RFRXpacket`, `getRSSI`

Example 1	`// Fast response receiving in STD mode`
	`_checkRFcfg_PQT = 1;  // Check RF signal for preamble quality as well`
	`if (checkRF(5))       // Detect signal with RSSI >= selected level`
	`{`
	`  if (RFRXpacket())   // Duration according to toutRF only if packet is sent.`
	`  {                   // toutRF can be optimized for expected packet length.`
	`    ...`
	`  }`
	`}              // Otherwise only ~2.8 ms (or 1 ms if _checkRFcfg_PQT = 0) is spent.`
	`... time-critical section can be placed here`

Example 2	`if (checkRF(10))    // Detect signal with RSSI >= selected level` `  ...`

Example 3	`// LP TX packet received in STD RX`
	`_checkRFcfg_PQT = 1;       // Check RF signal for preamble quality as well`
	`setRFmode(_RX_STD | _WPE); // Standard RX`
	`toutRF = 5;                // 5 or more is required, see Remarks`
	`                          // Change 5 to 100 if XLP packet is expected`
	`if (checkRF(5))            // Detect signal with RSSI >= selected level`
	`{`
	`  if (RFRXpacket())       // Duration according to toutRF only if packet is sent.`
	`    ...`

Example 4	`// RF signal strength analyzer`
	`while (1)`
	`  if (checkRF(5)) pulseLEDR(); // LED flash if RSSI >= selected level detected`

4.11.6 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 [5].
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. • To get the most precise RSSI value, use the strongest filter – `checkRF(90)`.
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 1	`if checkRF(0)`
	`{`
	`  if RFRXpacket()`
	`  {`
	`  ...`
	`  i = lastRSSI;  // Get current RSSI level. getRSSI is not needed.`
	`  }`
	`}`

Example 2	`checkRF(90);  // 90 gets more precise value than 0 but checking takes longer`
	`i = getRSSI();  // Get current RSSI level`
	`  ...`
	`checkRF(5);  // Then do not forget to use a reasonable checkRF filter`
	`  // before possible subsequent (X)LP receipt`

4.11.7 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 `setRFpower`
	• IQMESH:
	  • `PIN = 0x80` (IQMESH)
	  • Other network related parameters should also be specified
	• If User encryption is used, the packet length must be selected with respect to ciphertext length. See `encryptBufferRF`.
	See IQRF OS User's guide [1].

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 packet type and user data length. • `RFTXpacket` is allowed to be called at least 5 ms after `RFRXpacket`. See Example 4. • See Examples E01–TX, E03–TR and E09–LINK [9].
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`, `encryptBufferRF`, `setRFpower`, `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)
```

Example 4

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

4.11.8 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.
	• `_NTWF`: 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 [1], 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].
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. • See Examples E02–RX, E03–TR, E09–LINK~~, E11-IQMESH-DFM-N~~ and E14-CONSUMPTION [9].
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 answerSystemPacket

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
}
```

4.12 Networking

4.12.1 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	–

4.12.2 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	–

4.12.3 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`

4.12.4 setNetworkFilteringOn

Function	Start filtering incoming non-networking packets.
Purpose	To select whether to receive non-networking packets or not.
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 a networking packet came`

4.12.5 setNetworkFilteringOff

Function	Stop filtering incoming non-networking packets.
Purpose	To receive all packets (networking as well as non-networking).
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 a networking or non-networking`
	`// packet came.`

4.12.6 getNetworkParams

Function	Get network parameters
Purpose	Get some information about curent system, RF and network parameters
Syntax	`uns8` **`getNetworkParams()`**
Parameters	–
Return value	`userInterface` register. See IQRF OS User's guide [1], chapter *User interface*.

Output values	• `param2`: Address of the device in network
	  • 0 - Illegal value (resulting probably due to forbidden getNetworkParams usage at unbonded device)
	  • 1 – 239 Bonded Node (logical address)
	  • 254 (`0xFE`) Prebonded Node, not yet authorized
	• bit `_NTWF`
	  • 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 [1], Appendix 2, table OS, RF and network parameters.

Preconditions	• For IQMESH only. • For bonded devices only, see *Example*.
Remarks	–
Side effects	–
See also	`amIBonded`, `removeBondAddress`

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

4.12.7 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 `PCMD` 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 63 Nodes:
	  • For not selective FRC: from nodes with logical addresses 1-63
	  • For selective FRC: from up to 63 nodes selected from 239 Nodes
	  • 2B mode: 2 bytes from up to 31 Nodes:
	  • For not selective FRC: from nodes with logical addresses 1-31
	  • For selective FRC: from up to 31 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 (>63 b for 1B FRC or >31 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 1-63 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 … S63 mean up to 63 selected Nodes (selected from N1 to N239 by the bit array, see *Preconditions*).
	• In 2B mode, collected data (little endian) is stored at bytes 2-63 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 … S31 mean up to 31 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 Only bonded Nodes are allowed to be selected. • 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 `setFRCresponseTime`. It is the maximal time between finished routing and `responseFRC` calls, the same for all responding Nodes. It is selectable from 8 possible values (from 40 ms to 20.48 s) and passed to responding Nodes within the `sendFRC` command. See `responseFRC`, Example 1. Header file IQRF-macros.h defines all the 8 possible periods by constants `_FRC_RESPONSE_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 closest higher time `_FRC_RESPONSE_TIME_640_MS` should be used. • `clearBufferINFO` and `PIN = 0` must be performed first. See Examples.
Remarks	• See 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. `sendFRC` blocking time is: • Standard FRC: `BONDED_NODES * 30 + (DISCOVERED_NODES + 2) * 100 + _FRC_RESPONSE_TIME_xxx_MS + 210` [ms] • Advanced FRC and STD mode: `BONDED_NODES * 30 + (DISCOVERED_NODES + 2) * 110 + _FRC_RESPONSE_TIME_xxx_MS + 220` [ms] • Advanced FRC and LP mode: `BONDED_NODES * 30 + (DISCOVERED_NODES + 2) * 160 + _FRC_RESPONSE_TIME_xxx_MS + 260` [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 `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
setFRCresponseTime(_FRC_RESPONSE_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
setFRCresponseTime(_FRC_RESPONSE_TIME_360_MS);  // responseFRC must be called
                          // up to 360 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
setFRCresponseTime(_FRC_RESPONSE_TIME_680_MS); // responseFRC must be called
                         //up to 680 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
setFRCresponseTime(_FRC_RESPONSE_TIME_680_MS); // responseFRC must be called
                                  // up to 680 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));
```

4.12.8 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:
	  • `PNUM` contains the `__FRCOMMAND` (standard FRC) or `__FRCOMMANDADV` (advanced FRC) constant value.
	  • `PCMD` contains a user command sent from the Coordinator as the parameter of function `sendFRC`
	  Bit.7 in register `PCMD` 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 63 selected Nodes:
	  • For non-selective FRC: from Nodes with addresses 1-63
	  • For selective FRC: from up to 63 selected Nodes (selected from N1–N239 in bit array)
	  Flag `_twoByteFRC` 1:
	  2 bytes (`responseFRCvalue2B`) – from up to 31 selected Nodes:
	  • For non-selective FRC: from Nodes with addresses 1-31
	  • For selective FRC: from up to 31 selected Nodes (selected from N1–N239 in bit array)
	  • `param4` contains the time (in ticks) specified on the Coordinator side by macro `setFRCresponseTime`. 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 `setFRCresponseTime`). 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 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 `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
                            // setFRCresponseTime 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 (PNUM == __FRCOMMAND)
          {
             uns16 user_value;
             user_value.low8 = DataOutBeforeResponseFRC[0];
             user_value.high8 = DataOutBeforeResponseFRC[1];
                    // 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 PCMD command (and possibly
              // according to DataOutBeforeResponseFRC), e.g. myResponse=…;
              // Shaded part must take up to the time specified in param4
          if (PCMD.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
                            // setFRCresponseTime macro on the Coordinator side

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

4.12.9 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 (for selective as well as 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 `responseFRC`.
Side effects	–
See also	`responseFRC`
Example	See `responseFRC` Example.

4.13 Routing

4.13.1 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`, `isDiscoveredNode`, `wasRouted`
Example	–

4.13.2 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`, `isDiscoveredNode`, `wasRouted`
Example	–

4.13.3 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 RF 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 IQRF OS User's guide [1], 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 `RTHOPS`, `RTTSLOT`, … may be changed. • A/D converter control registers are changed
See also	`setRoutingOn`, `setRoutingOff`, `isDiscoveredNode`, `bondNewNode`, `answerSystemPacket`

Example 1	`setRFpower(DISCOVERY_POWER);  // Set RF TX 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`
	`setRFpower(MY_POWER);  // Restore RF power`

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

4.13.4 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 RF 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	• RF 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 `RTHOPS`, `RTTSLOT`, … may be changed. • A/D converter control registers are modified.
See also	`setRoutingOn`, `setRoutingOff`, `isDiscoveredNode`, `discovery`, `nodeAuthorization`

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

4.13.5 isDiscoveredNode

Function	Check for being discovered
Purpose	Ask whether the Node has been discovered
Syntax	`bit` **`isDiscoveredNode(address)`**
Parameters	`uns8 address`: Node address (1 to 239)
Return value	• `true`: Specified Node has been discovered • `false`: Specified Node has not been discovered
Output values	–
Preconditions	For IQMESH Coordinator only.
Remarks	–
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`
	`  …`
	`}`

4.13.6 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.
Side effects	–
See also	`setRoutingOn`, `setRoutingOff`, `discovery`, `isDiscoveredNode`

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

4.13.7 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 discovered 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 IQRF OS User's guide.
Side effects	–
See also	`discovery`, `isDiscoveredNode`

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

4.14 Bonding – Node only

4.14.1 bondRequestAdvanced

Function	Ask Coordinator for bonding or other Node 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 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 4 B user data is exchanged between prebonded device and the device providing prebonding.
Syntax	`bit` **`bondRequestAdvanced()`**
Parameters	–
Input values	`userBondingData[4]` - user data to be delivered to the Node or Coordinator providing prebonding.
Return value	• 1 – Node has been bonded or prebonded • 0 – Node has neither been bonded nor prebonded
Output values	• The `amIBonded` function starts to return the value `= 1` whenever is called while the Node is bonded by `bondRequestAdvanced` and not beeing unbonded by `removeBond`. • `userBondingData[4]` - user data delivered from the Node or Coordinator providing prebonding. • 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`, `prebondNodeAtNode`, `prebondNodeAtCoordinator` and IQRF User's guide, chapter *Bonding*. The counter is cleared after reset.
Preconditions	• The same Access password must be set (in TR configuration or by `setAccessPassword`) as at the Coordinator.
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 check the result later on by `amIBonded` and possibly change it by `removeBond` or `removeBondAddress`. • Prebonding is an initial phase of remote bonding. The new (bond requesting) Node gets the network ID, the universal address `0xFE` and Network password from another (already bonded) Node or the Coordinator. Prebonded Node becomes accessible RX only 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. • 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`, `prebondNodeAtNode`, `prebondNodeAtCoordinator`, `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();`
	`  // Data to be delivered to the prebonding device`
	`  UserBondingData[0] = myDataToPrebondingDevice[0];`
	`  ...`
	`  UserBondingData[3] = myDataToPrebondingDevice[3];`
	`  if (bondRequestAdvanced()) // Repeatedly try to bond`
	`  {`
	`  pulseLEDR();`
	`  // Data received from the prebonding device`
	`  myDataFromPrebondingDevice[0] = UserBondingData[0];`
	`  ...`
	`  myDataFromPrebondingDevice[3] = UserBondingData[3];`
	`  }`
	`  waitDelay(1);`
	`}  // Until successful`
	`setNodeMode();  // Restore`

4.14.2 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	–
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`

4.14.3 removeBondAddress

Function	Change logical Node address to the universal one (`0xFE`). `NID` and Network password stay unchanged, therefore the Node still stays in the network.
Purpose	E.g. to enable subsequent change of the Node address by reauthorization.
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 reauthorization, it is recommended to save the `MID` of given Node by the application first.
Side effects	–
See also	`amIBonded`, `getNetworkParams`, `nodeAuthorization`
Example	`removeBondAddress();  // Change current logical address to universal address`

4.14.4 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	–
Remarks	• 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`

4.14.5 prebondNodeAtNode

Function	Look for bond requesting devices and prebond a new Node based on its request via RF by another (already bonded) Node. Assign the universal address `0xFE` and send it together with Network ID and Network password 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. It operates without direct action of the Coordinator. Then it is needed to deliver the obtained `MID` to the Coordinator (the second phase) by application program. To complete remote bonding, the Coordinator must authorize the new Node (the third phase of the remote bonding process). Moreover, a 4 B user data is exchanged between prebonded device and the device providing prebonding.
Syntax	`bit` **`prebondNodeAtNode()`**
Parameters	–
Input values	• `userBondingData[4]` - user data to be delivered to prebonded Node
Return value	• 1 – prebonding successful. `bufferINFO` contains the MID of prebonded Node • 0 – prebonding unsuccessful. `bufferINFO` unchanged
Output values	• `bufferINFO[0 to 3]`: MID of the prebonded Node (if successfully prebonded), LSB in `bufferINFO[0]`. • `userBondingData[4]`- user data delivered from the prebonded Node within the communication during prebonding. This data is valid after `RFRXpacket()` if `networkInfo[17] = 0x01`. This feature is intended for OS and DPA internal purposes rather than for common usage. See Example.
Preconditions	• For IQMESH Node only. (`setNodeMode` must be called first.) • The same Access password as at the Coordinator must be set (in TR configuration or by `setAccessPassword`) at prebonded Node as well as at the Node providing prebonding. • This function can even be called in Coordinator mode but the device is automatically switched into Node mode. • The Node providing prebonding must already be bonded in given network (to be able to provide the Network ID and Network password) and must be accessible from the Coordinator (either directly or via other Nodes). • While the new Node is requesting to be prebonded (by function `bondRequestAdvanced`), the `prebondNodeAtNode` function must be called in RF RX loop. See the Example below. • To avoid possible prebonding of a device not intended for given network, it is recommended to have bonding password changed from its default value.
Remarks	• See 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. Long running 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 [1], 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 `prebondNodeAtNode` • A/D converter control registers are modified
See also	`prebondNodeAtCoordinator`, `bondRequestAdvanced`, `setNodeMode`, `setCoordinatorMode`, `nodeAuthorization`, `answerSystemPacket`

Example

```
while(1)
{
  if (RFRXpacket())
  {
    ...
  }
  else
  {
    if (prebondingEnabled && (networkInfo[17] == 0x01))
                    // This feature is intended for OS and DPA internal purposes
                    // rather than for common usage.
    {
                         // Store data from prebonded Node
      myDataFromPrebondedNode[0] = UserBondingData[0];
           ...
      myDataFromPrebondedNode[3] = UserBondingData[3];

                         // Data to be delivered to prebonded Node
      UserBondingData[0] = myDataToPrebondedNode[0];
           ...
      UserBondingData[3] = myDataToPrebondedNode[3];

                        // Try prebonding if enabled by user
      if (prebondNodeAtNode())
      {
        setNodeMode();  // Restore
        ...             // MID of prebonded Node is stored in bufferINFO

      }
     ...
    }
  }
  ...
}
```

4.15 Bonding – Coordinator only

4.15.1 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 send it together with Network ID and Network password 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` • 0 The lowest free address is assigned. The block of occupied addresses need not be continuous, possible vacations are allowed. In case of a discontinuity, the lowest vacation will be occupied. • 1 to 239 Assign requested address to the Node. This must be unique in the whole network. Only these Nodes can belong to routing backbone. • 254 (`0xFE`) The universal address. Nodes with this address are included in the network but outside the routing backbone (not routing and not being discovered).

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:
	  • 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.
	  • Return after 10 s. No device is requesting to be bonded.

Output values	• `param2`: Node number • `BondingNodeMID`: MID of the bonded Node. • 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.) • The same Access password must be set at the Node (in TR configuration or by `setAccessPassword`) as at the Coordinator. • 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 for bonding in STD mode only. For bonding in power saving LP mode, use the same procedure as for remote bonding (using prebondNodeAtCoordinator and nodeAuthorization) instead.
Remarks	• See 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 MID 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 `prebondNodeAtNode` and `prebondNodeAtCoordinator` *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, setCoordinatorMode, nodeAuthorization, prebondNodeAtNode, prebondNodeAtCoordinator

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

4.15.2 prebondNodeAtCoordinator

Function	Look for bond requesting devices and prebond or bond a new Node based on its request via RF by the Coordinator. Assign a Node address and send it together with Network ID and Network password 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. To complete remote bonding, the Coordinator must authorize the new Node not being prebonded for a particular address (the final phase of the remote bonding process). If prebonded for a particular address, the authorization is not needed. Moreover, a 4 B user data is exchanged between prebonded device and the Coordinator.
Syntax	`bit` **`prebondNodeAtCoordinator(address)`**
Parameters	`uns8: address` • 0 The lowest free address is assigned. The block of occupied addresses need not be continuous, possible vacations are allowed. In case of a discontinuity, the lowest vacation will be occupied. • 1 to 239 Assign requested address to the Node. This must be unique in the whole network. Only these Nodes can belong to routing backbone. The result (the Node is bonded without prebonding) is the same as when `bondNewNode` is used. • 254 (`0xFE`) The universal address. Nodes with this address are included in the network but outside the routing backbone (not routing and not being discovered).
Input values	• `userBondingData[4]` - user data to be delivered to prebonded Node
Return value	• 1 – prebonding successful. `bufferINFO` contains MID of prebonded Node • 0 – prebonding unsuccessful. `bufferINFO` unchanged
Output values	• `bufferINFO[0` to `3]`: MID of the prebondedNode (if successfully prebonded), LSB in `bufferINFO[0]`. • `userBondingData[4]`- user data delivered from the prebonded Node within the communication during bonding. This data is valid after `RFRXpacket()` if `networkInfo[17] = 0x01`. This feature is intended for OS and DPA internal purposes rather than for common usage. See Example.
Preconditions	• For IQMESH Coordinator only. (`setCoordinatorMode` must be called first.) • This function can even be called in Node mode but the device is automatically switched into Coordinator mode. • 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. • The Node must be directly accessible from the Coordinator. • The same Access password must be set at the Node (in TR configuration or by `setAccessPassword`) as at the Coordinator. • While the new Node is requesting to be prebonded (by function `bondRequestAdvanced`), the `prebondNodeAtCoordinator` function must be called in RF RX loop. See the Example below. • To avoid possible prebonding of a device not intended for given network, it is recommended to have bonding password changed from its default value.
Remarks	• See 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. Long running can ask for troubles with unwanted prebonding. • 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 [1], 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 prebondNodeAtCoordinator • A/D converter control registers are modified
See also	prebondNodeAtNode, bondRequestAdvanced, setNodeMode, setCoordinatorMode, nodeAuthorization, answerSystemPacket
Example	

```
while(1)
{
  if (RFRXpacket())
  {
    ...
  }
  else
  {
    if (prebondingEnabled && (networkInfo[17] == 0x01))
                    // This feature is intended for OS and DPA internal purposes
                    // rather than for common usage.
    {
                        // Store data from prebonded Node
      myDataFromPrebondedNode[0] = UserBondingData[0];
           ...
      myDataFromPrebondedNode[3] = UserBondingData[3];

                        // Data to be delivered to prebonded Node
      UserBondingData[0] = myDataToPrebondedNode[0];
           ...
      UserBondingData[3] = myDataToPrebondedNode[3];

                       // Try prebonding if enabled by user
      if (prebondNodeAtCoordinator(0xFE))
      {
        setNodeMode();  // Restore
        ...             // MID of prebonded Node is stored in bufferINFO

      }
      ...
    }
  }
  ...
}
```

4.15.3 nodeAuthorization

Function	Authorize or refuse authorization for the prebonded Node. To be authorized, 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. To refuse authorization, unbond and restart an unwanted Node.
Purpose	• Include a new (prebonded) Node to the network. A last phase of the remote bonding process. • Exclude an unwanted device attempting to be bonded.
Syntax	`bit` **`nodeAuthorization(address)`**
Parameters	`uns8 address`: • 0 The lowest free address is assigned. The block of occupied addresses need not be continuous, possible vacations are allowed. In case of a discontinuity, the lowest vacation will be occupied. • 1 to 239 Assign requested address to the prebonded Node. This must be unique in the whole network. Only Nodes in this range can be parts of routing backbone. • 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. It is intended especially for networks with more than 239 Nodes. • 0xFF Prebonded Node is unbonded and reset.
Input values	The `MID` of prebonded Node to be authorized must be stored in `BondingNodeMID[4]`, MSB first. See Example.

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. Possible reasons:
	  • 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 Authorization. • If the Node has been prebonded at the Coordinator for a particular address, this authorization is not needed.
Remarks	• See IQRF OS User's guide, chapter *Bonding*. • This function sends an authorization packet (broadcast), waits for delivery and provides appropriate checking. It is a blocking function (not running in OS background) until routing of the authorization packet to destination Node is finished. • An occupied address can be unblocked by `removeBondedNode(address)`. • RF power and RF channel is not affected. The STD mode is used. • The authorization packet is sent without routing which significantly speeds up the authorization.
Side effects	`PIN`, `DLEN`, `bufferRF` and `bufferINFO` are modified
See also	`bondNewNode`, `prebondNodeAtNode`, `prebondNodeAtCoordinator`, `removeBondedNode`, `isBondedNode`, `setCoordinatorMode`, `bondRequestAdvanced`, `answerSystemPacket`, `removeBondAddress`

Example	`BondingNodeMID[0] = MID_3;`
	`  ...`
	`BondingNodeMID[3] = MID_0;`
	
	`if (nodeAuthorization(mid, address))  // Was the address free?`
	`{  // Yes:`
	`  ...  // 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`

4.15.4 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(address)`**
Parameters	`uns8 address`: Node number

Return value	• For Nodes from `1` to `0xEF`:
	  • `1` – Node is in the list of bonded Nodes
	  • `0` – Node is not in the list of bonded Nodes
	• For Nodes from `0xF0` to `0xFD`: `0`
	• For Nodes from `0xFE` to `0xFF`: `1`

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`
	`}`

4.15.5 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(address)`**
Parameters	`uns8 address`: 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`

4.15.6 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(address)`**
Parameters	`uns8 address`: 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`

4.15.7 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	• 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`

4.16 Encryption

4.16.1 setAccessPassword

Function	Set Access password
Purpose	To specify the 16 B long password for generating keys for encryption/decryption of bonding and maintenance (e.g. authorization and Restore) communication
Syntax	`void` **`setAccessPassword()`**
Parameters	–
Input values	`bufferINFO[0` to `15]` to be copied to Access password.
Return value	–
Output values	Complete `bufferINFO` is cleared when finished.
Preconditions	• For IQMESH only • Default value after reset: Access password = `00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00` • It is recommended to change Access password from default value to a user-specific one. • The same Access password must be used for all devices in the network.
Remarks	AES-128 CBC encryption/decryption is used.
Side effects	–
See also	–

Example	`bufferINFO[0] = 0x52;`
	`…`
	`bufferINFO[15] = 0xB1;`
	`setAccessPassword();`

4.16.2 setUserKey

Function	Set user encryption/decryption key for RF communication
Purpose	To specify the 16 B long key for user-specific encryption and decryption
Syntax	`void` **`setUserKey()`**
Parameters	–
Input values	`bufferINFO[0` to `15]` to be copied to User key.
Return value	–
Output values	Complete `bufferINFO` is cleared when finished.
Preconditions	• Default value after reset: User key = `00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00` • For networking as well as non-networking communication
Remarks	AES-128 CBC encryption/decryption is used.
Side effects	–
See also	`encryptBufferRF`, `decryptBufferRF`

Example	`bufferINFO[0] = 0x52;`
	`…`
	`bufferINFO[15] = 0xB1;`
	`setUserKey();`

4.16.3 encryptBufferRF

Function	Encrypt `bufferRF`
Purpose	Payload data encryption by the user-specific User key
Syntax	`void` **`encryptBufferRF(blocks)`**
Parameters	`uns8 blocks`: Number of 16 B blocks to be encrypted (1 to 4)
Return value	–
Output values	Specified number of blocks in `bufferRF` is encrypted by the User key.
Preconditions	• For networking as well as non-networking communication. • It is not allowed to send an encrypted 16 B block incomplete otherwise it can not be decrypted correctly. • All encrypted blocks must completely be sent otherwise the whole plaintext can not be decrypted correctly.
Remarks	• If `blocks < 4`, the rest of `bufferRF` remains unencrypted. • Industry standard AES-128 b CBC, `IV = 00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00` encryption is used.
Side effects	–
See also	`setUserKey`, `decryptBufferRF`

Example	`// When User key and bufferRF content are already prepared`
	`encryptBufferRF(2);  // 32 B encrypted`
	`// Number of bytes to be sent:`
	`DLEN = 56;  // 56 B: correct (incomplete 24 B block is not encrypted)`
	`DLEN = 32;  // 32 B: correct`
	`DLEN = 16;  // 16 B: correct (but probably not practical)`
	`DLEN = 16;  // 16 B: not correct (incomplete encrypted block sent)`
	`DLEN = 31;  // 31 B: not correct (incomplete encrypted block sent)`
	`RFTXpacket();`

4.16.4 decryptBufferRF

Function	Decrypt `bufferRF`
Purpose	Decryption of payload data encrypted by `encryptBufferRF`
Syntax	`void` **`decryptBufferRF(blocks)`**
Parameters	`uns8 blocks`: Number of 16 B blocks to be decrypted (1 to 4)
Return value	–
Output values	Specified number of blocks in `bufferRF` is decrypted by the User key.
Preconditions	• The same User key must be used as for preceding encryption. • For networking as well as non-networking communication. • It is not necessary to decrypt data within IQRF wireless. Decryption can alternatively be done e.g. by a superordinate system.
Remarks	• If `blocks < 4`, the rest of `bufferRF` remains unchanged. • Industry standard AES-128 b CBC, `IV = 00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00` decryption is used.
Side effects	–
See also	`setUserKey`, `encryptBufferRF`

Example	`bufferINFO[0 to 15] = ...;`
	`setUserKey();  // The same as for encryption`
	`…`
	`if (RFRXpacket())`
	`{`
	`  decryptBufferRF(2);  // 32 B decrypted, the rest remains unchanged`
	`  ...`

	}

4.17 RFPGM – wireless upload

4.17.1 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 [8].
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`

4.17.2 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 [8].
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`

4.17.3 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 [8] 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 [1], 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`

4.17.4 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	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 [8])

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. (This bit operates like `enableRFPGM` / `disableRFPGM` functions.)
- H – enabled
- L – disabled (default).

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

Bit 7: RFPGM termination by MCU pin 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 non-SMT TR modules, e.g. TR-72D
  - Q12 for SMT TR modules, e.g.TR-76D

This time must be prolonged up to 2 s in case of strong RF noise.

Bits 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 pin is enabled, pin RB4 must have a pull-up resistor. `RB4` has a SW selectable internal pull-up, default enabled by OS after boot.

Remarks
- RFPGM invoking by `runRFPGM` is unconditional, independent on parameter `x`
- RFPGM termination by IQRF IDE [8] 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`

Example 4	`setupRFPGM(_ENABLE_ON_RESET | _DUAL_CHANNEL);`
	`// The same RFPGM setup as in Example 1 but using predefined constants.`
	`// See chapter Macros / Constants.`

5 Macros

Macros described below are intended for better mnemonic and compatibility with older versions. They are included in the `IQRF-macros.h` header file provided with other header files dedicated to given TR transceiver and IQRF OS version. It is not recommended to make any changes in it. When needed, the user should create another header file with his own macros.

5.1 Constants

Name	Interpretation	Description
`TRUE`	`1`	An alternative for C language
`FALSE`	`0`	An alternative for C language
`F_OSC`	`16000000`	16 MHz MCU clock. Refer to IQRF OS User's guide [1], *Oscillator* in chapter *Microcontroller*.
`TX_POWER_MAX`	`7`	Maximal RF output power level (specified by `setRFpower(level)`)
`EEE_BLOCK_SIZE`	`16`	External EEPROM data block size
For `setRFmode(mode)`		
`_RX_STD`	`0x00`	RX mode STD
`_STDL`	`0x80`	Prolong preamble for STD TX mode
`_RX_LP`	`0x01`	RX mode LP
`_RX_XLP`	`0x02`	RX mode XLP
`_TX_STD`	`0x00`	TX mode STD
`_TX_LP`	`0x10`	TX mode LP
`_TX_XLP`	`0x20`	TX mode XLP
`_RLPMAT`	`0x04`	LP/XLP RX asynchronous termination
`_WPE`	`0x40`	Wait Packet End
Example: `setRFmode(_RX_STD | _TX_STD | _STDL | _WPE);` // STD RX, STD TX, preamble ~8 ms selected, Wait Packed End enabled		
For `setupRFPGM(x)`		
`_DUAL_CHANNEL`	`0x03`	RFPGM dual channel receiving
`_LP_MODE`	`0x04`	RFPGM low power mode receiving
`_ENABLE_ON_RESET`	`0x10`	RFPGM invoking by reset
`_TIME_TERMINATE`	`0x40`	RFPGM auto termination after ~1 min
`_PIN_TERMINATE`	`0x80`	RFPGM termination by MCU pins

5.2 Control

5.2.1 reset

Macro	Reset MCU
Purpose	Restart MCU, IQRF OS and application SW from very beginning
Syntax	• void reset() • Alternative `softReset()` is also possible
Parameters	–
Return value	–
Output values	MCU SW reset
Preconditions	–
Remarks	• This macro is equivalent to MCU machine instruction `Reset` and CC5X command `softReset()`. • This SW reset slightly differs in initialization from other reset types (power-on, watchdog and BOR). See respective MCU datasheet [6].
Side effects	–
See also	`wasRFICrestarted`

Example	`if (...)  // When specified condition met`
	`  reset();  //  Invoke MCU software reset`
	`  ...  // Otherwise continue`

5.2.2 setBORon

Macro	Enable MCU Brown-out reset (BOR)
Purpose	To enable MCU reset automatically when power supply falls below 1.9 V for 3 µs (typical values)
Syntax	`void setBORon()`
Parameters	–
Return value	–
Output values	BOR enabled
Preconditions	BOR is default disabled after power on.
Remarks	• Refer to the datasheet of given TR module [4] and IQRF OS User's guide, chapter *Reset*. • To minimize power consumption, BOR should be disabled before entering Sleep or Deep sleep.
Side effects	–
See also	`setBORoff`
Example	See `setBORoff`

5.2.3 setBORoff

Macro	Disable MCU Brown-out reset (BOR)
Purpose	To disable BOR, e.g. to reduce power consumption before sleep
Syntax	`void setBORoff()`
Parameters	–
Return value	–
Output values	BOR disabled
Preconditions	BOR is default disabled after power on.
Remarks	• Refer to the datasheet of given TR module [4] and IQRF OS User's guide, chapter *Reset*. • To minimize power consumption, BOR should be disabled before entering Sleep or Deep sleep.
Side effects	–
See also	`setBORon`

Example	`setBORon();  // Enable BOR at beginning`
	`...`
	`setBORoff;  // Disable BOR before sleep`
	`iqrfSleep();  // Sleep`
	`setBORon();  // Reenable BOR after wake-up`
	`...`

5.2.4 setWDTon

Macro	Enable Watchdog
Purpose	Enable Watchdog (to increase the reliability or to enable wake-up from sleep on watchdog timeout)
Syntax	`void setWDTon()`
Parameters	–
Return value	–
Output values	MCU Watchdog enabled
Preconditions	Watchdog is default disabled and its timeout is set to 4 s after power on.
Remarks	Refer to respective MCU datasheet [6] and IQRF OS User's guide, chapter Watchdog.
Side effects	–
See also	`setWDToff`, `setWDTon_xxxx`

Example	`setWDTon;  // Watchdog enabled`
	`iqrfSleep();  // Sleep, wake-up after default 4 s (unless otherwise stated)`
	`setWDToff();  // Continue, Watchdog disabled`

5.2.5 setWDToff

Macro	Disable Watchdog
Purpose	When disabled, no Watchdog timeout is generated and wake-up from sleep on watchdog timeout is disabled.
Syntax	`void setWDToff()`
Parameters	–
Return value	–
Output values	Watchdog disabled
Preconditions	Watchdog is default disabled and its timeout is set to 4 s after power on.
Remarks	Refer to respective MCU datasheet [6] and IQRF OS User's guide, chapter Watchdog.
Side effects	–
See also	`setWDTon`, `setWDTon_xxxx`
Example	See `setWDTon`

5.2.6 setWDTon_xxxx

Macro	Enable Watchdog with wake-up after specifid time
Purpose	Specify a Watchdog timeout (e.g. to define the sleeping period)

Syntax	`void setWDTon_1ms()`
	`void setWDTon_2ms()`
	`void setWDTon_4ms()`
	`void setWDTon_8ms()`
	`void setWDTon_16ms()`
	`void setWDTon_32ms()`
	`void setWDTon_64ms()`
	`void setWDTon_128ms()`
	`void setWDTon_256ms()`
	`void setWDTon_512ms()`
	`void setWDTon_1s()`
	`void setWDTon_2s()`
	`void setWDTon_4s()`
	`void setWDTon_8s()`
	`void setWDTon_16s()`
	`void setWDTon_32s()`
	`void setWDTon_64s()`
	`void setWDTon_128s()`
	`void setWDTon_256s()`

Parameters	–
Return value	–
Output values	Watchdog is enabled and its timeout configured for specified time (1 ms, …, 256 s)
Preconditions	Watchdog is default disabled and its timeout is set to 4 s after power on.
Remarks	Refer to the datasheet of given TR module [4] and IQRF OS User's guide, chapter Watchdog.
Side effects	–
See also	`setWDTon`, `setWDToff`

Example	`setWDTon_16s();  // Watchdog enabled`
	`iqrfSleep();  // Sleep, wake-up after 16 s`
	`setWDToff();  // Continue, Watchdog disabled`

5.2.7 sleepWOC

Macro	TR Sleep with wake-up on change at dedicated TR pin enabled
Purpose	Put TR into power saving mode and enable wake-up on external event
Syntax	`void sleepWOC()`
Parameters	–
Return value	–
Output values	TR sleeping and waiting for pin change
Preconditions	The same as for `iqrfSleep`
Remarks	• Wake-up can be caused on C5 (for TR modules for SIM mounting, e.g. TR-72D) or Q12 (for SMT mounting, e.g.TR-76D) pin change. • Both rising and falling edge on the pin is active. The macro can easily be modified in source code for only one of these edges.
Side effects	• MCU watchdog is disabled and not reenabled after wake-up. • MCU global interrupt is enabled after wake-up. • MCU register `FSR1` is destroyed
See also	`iqrfSleep`, `iqrfDeepSleep`, `buttonPressed`

Example	`stopLEDR();  // Disable all power consuming HW resources under user's control`
	`sleepWOC();  // Sleep with wake-up on pin change`
	`if (buttonPressed) // If button is pressed`
	`{`
	`  pulseLEDR();  //  Indicate wake-up by red LED`
	`  ...  //  and continue`
	`}`

5.2.8 setIOCBN

Macro	Set the MCU flag `IOCBN4`.
Purpose	To configure interrupt on pin change to detect falling edge.
Syntax	`void setIOCBN()`
Parameters	–
Return value	–
Output values	Flag `IOCBN4` is set.
Preconditions	–
Remarks	• This macro works with MCU pin RB4. It is the dedicated MCU pin for interrupt on change at TR transceivers. It is connected to TR pin C5 (for TRs for SIM mounting, e.g. TR-72D) or Q12 (for TRs for SMT mounting, e.g.TR-76D) . • IQRF development tools (e.g. CK-USB-04A and DK-EVAL-04A) with a TR module for SIM mounting, e.g. TR-72D (but not with a TR module for SMT mounting, e.g. TR-76D) use this pin to connect the User pushbutton (SW1), active low. See respective PIC datasheet [6] and IQRF OS User's guide [1], chapters *MCU pins* and *Interrupt*.
Side effects	–
See also	`clearIOCBN`, `clearIOCF`
Example	See `clearIOCF`.

5.2.9 clearIOCBN

Macro	Clear the MCU flag `IOCBN4`.
Purpose	To configure interrupt on pin change not to detect falling edge.
Syntax	`void clearIOCBN()`
Parameters	–
Return value	–
Output values	Flag `IOCBN4` is cleared.
Preconditions	–
Remarks	• This macro works with MCU pin RB4. It is the dedicated MCU pin for interrupt on change at TR transceivers. It is connected to TR pin C5 (for TRs for SIM mounting, e.g. TR-72D) or Q12 (for TRs for SMT mounting, e.g.TR-76D) . • IQRF development tools (e.g. CK-USB-04A and DK-EVAL-04A) with a TR module for SIM mounting, e.g. TR-72D (but not with a TR module for SMT mounting, e.g. TR-76D) use this pin to connect the User pushbutton (SW1), active low. See respective PIC datasheet [6] and IQRF OS User's guide [1], chapters *MCU pins* and *Interrupt*.
Side effects	–
See also	`setIOCBN`, `clearIOCF`
Example	

```
setIOCBN();          // Falling edge active
   ...
if (IOCBF.4)         // Falling edge detected?
{
  pulseLEDR();       // Yes, perform desired service
  clearIOCF();       //   and clear interrupt on pin change flag
}
   ...
clearIOCBN();        // Falling edge not active
IOCBP.4 = 1;         // Rising edge active
   ...
if (IOCBF.4)         // Rising edge detected?
{
  pulseLEDG();       // Yes, perform desired service
  clearIOCF();       //   and clear interrupt on pin change flag
}
   ...
```

5.2.10 clearIOCF

Macro	Clear the MCU interrupt on pin change flag `IOCBF4`.
Purpose	`IOCBF4` is a flag informing that specified condition for interrupt on pin change has occurred. Once this event is serviced, the flag must be cleared to avoid recursive interrupts.
Syntax	`void clearIOCF()`
Parameters	–
Return value	–
Output values	Flag `IOCBF4` is cleared.
Preconditions	• This macro must be called (often in interrupt service routine) before re-enabling interrupts. • The pin change can also be serviced by polling of this flag (without an interrupt).
Remarks	• This macro works with MCU pin RB4. It is the dedicated MCU pin for interrupt on change at TR transceivers. It is connected to TR pin C5 (for TRs for SIM mounting, e.g. TR-72D) or Q12 (for TRs for SMT mounting, e.g.TR-76D). • IQRF development tools (e.g. CK-USB-04A and DK-EVAL-04A) with a TR module for SIM mounting, e.g. TR-72D (but not with a TR module for SMT mounting, e.g. TR-76D) use this pin to connect the User pushbutton (SW1), active low. • See respective PIC datasheet [6] and IQRF OS User's guide [1], chapters *MCU pins* and *Interrupt*.
Side effects	–
See also	`setIOCBN`, `clearIOCBN`
Example	See `clearIOCBN`

5.2.11 breakpoint

Macro	Call `debug` with specified value in `w` register (the MCU accumulator)
Purpose	To identify given breakpoint via the `w` value
Syntax	`void breakpoint(wValue)` Alternative syntax `void debugW(wValue)` is also possible
Parameters	`uns8 wValue`: Value to be put into w register
Return value	–
Output values	• W = wValue • `debug` called
Preconditions	–
Remarks	Corresponding `wValue` is displayed in IQRF IDE when a breakpoint is reached.
Side effects	–
See also	`debug`

Example	`if(!eeeReadData(0x000))  // External EEPROM test`
	`{`
	`  breakpoint(1);  // Read unsuccessful`
	`}`
	`else`
	`{`
	`  breakpoint(2);  // Read successful`
	`}`

5.2.12 buttonPressed

Macro	Read the level at the pin dedicated to be checked
Purpose	Simple pin level checking (e.g. whether the pushbutton connected to this pin is pressed)
Syntax	`bit buttonPressed`
Parameters	–
Return value	• `true` If log.0 is detected on the pin • `false` If log.1 is detected on the pin
Output values	–
Preconditions	The dedicated pin must be configured as input. It is arranged in OS by default. OS itself never switch this pin to output.
Remarks	• The dedicated pin is C5 (for TR modules for SIM mounting, e.g. TR-72D) or Q12 (for SMT mounting, e.g.TR-76D). • It is connected to MCU pin RB4. Interrupt on change and wake-up from sleep can be utilized on this pin. • IQRF development tools (e.g. CK-USB-04A and DK-EVAL-04A) with a TR module for SIM mounting, e.g. TR-72D (but not with a TR module for SMT mounting, e.g. TR-76D) use this pin to connect the User pushbutton (SW1), active low.
Side effects	–
See also	–

Example 1	`if (buttonPressed)  //  If button is pressed`
	`{`
	`  pulseLEDR();  //  LED indication`
	`  ...`
	`}`

Example 2	`TRISB.4 = 1;  // Configure the pin as input. Required only if`
	`// previously changed by the user.`
	`// See IQRF User's guide, chapter MCU pins.`
	`if (buttonPressed)  //  If button is pressed`
	`  ...`

5.3 Serial EEPROM and temperature sensor

5.3.1 eEEPROM_TempSensorOn

Macro	Enable serial EEPROM and temperature sensor
Purpose	To switch serial EEPROM and temperature sensor on only when it is required with respect to power consumption
Syntax	`void eEEPROM_TempSensorOn()`
Parameters	–
Return value	–
Output values	Serial EEPROM and temperature sensor are connected to power supply
Preconditions	The default state after power on is On.
Remarks	• Both serial EEPROM and temperature sensor can be enabled at the same time only • To get temperature sensor ready, a delay is required after `eEEPROM_TempSensorOn`. See `getTemperature`.
Side effects	–
See also	`eEEPROM_TempSensorOff`
Example	See `eEEPROM_TempSensorOff`

5.3.2 eEEPROM_TempSensorOff

Macro	Disable serial EEPROM and temperature sensor
Purpose	To switch serial EEPROM and temperature sensor off to reduce power consumption
Syntax	`void eEEPROM_TempSensorOff()`
Parameters	–
Return value	–
Output values	Serial EEPROM and temperature sensor are disconnected from power supply
Preconditions	Because OS uses serial EEPROM to store some networking information, e.g. during Discovery, it is recommended to utilize such power management for non-networking applications only.
Remarks	
Side effects	–
See also	`eEEPROM_TempSensorOn`

Example	`eEEPROM_TempSensorOn();`
	`waitDelay(30);` `// 300 ms delay required`
	`getTemperature();`
	`eEEPROM_TempSensorOff();` `// Recommended in non-networking applications only`
	`temperature = param3;`

5.4 RAM

5.4.1 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 [9].
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);`

5.4.2 setFSR0

Function	Set control register FSR0 to access the beginning of specified OS buffer via indirect addressing
Purpose	
Syntax	`uns8` **`setFSR0(buffer)`**

Parameters	`buffer:` `_FSR_NINFO` Set `FSR` to `networkInfo`
	`_FSR_INFO` Set `FSR` to `bufferINFO`
	`_FSR_COM` Set `FSR` to `bufferCOM`
	`_FSR_AUX` Set `FSR` to `bufferAUX`
	`_FSR_RF` Set `FSR` to `bufferRF`
	`_FSR_ntwADDR` Set `FSR` to `ntwADDR`

Return value	64 Constant value to optimize possible subsequent work with buffers
Output values	• `FSR0` addresses byte[0] of specified OS buffer • `WREG = 64`
Preconditions	–
Remarks	See IQRF OS User's guide [1], chapter *Data memory (RAM)*.
Side effects	–
See also	`setFSR1`, `setFSR01`

Example	`setFSR0(_FSR_COM);  // FSR0 addresses bufferCOM[0]`
	`X = INDF0;  // X = bufferCOM[0]`

5.4.3 setFSR1

Function	Set control register FSR1 to access the beginning of specified OS buffer via indirect addressing
Purpose	
Syntax	`uns8` **`setFSR1(buffer)`**

Parameters	`buffer`: `_FSR_NINFO` Set `FSR` to `networkInfo`
	`_FSR_INFO` Set `FSR` to `bufferINFO`
	`_FSR_COM` Set `FSR` to `bufferCOM`
	`_FSR_AUX` Set `FSR` to `bufferAUX`
	`_FSR_RF` Set `FSR` to `bufferRF`
	`_FSR_ntwADDR` Set `FSR` to `ntwADDR`

Return value	64 Constant value to optimize possible subsequent work with buffers
Output values	• `FSR1` addresses byte[0] of specified OS buffer • `WREG = 64`
Preconditions	–
Remarks	See IQRF OS User's guide [1], chapter *Data memory (RAM)*.
Side effects	–
See also	`setFSR0`, `setFSR01`

Example	`setFSR1(_FSR_RF);  // FSR1 addresses bufferRF[0]`
	`X = INDF1;  // X = bufferRF[0]`

5.4.4 setFSR01

Function	Set control registers FSR0 and FSR1 to access the beginning of specified OS buffers via indirect addressing
Purpose	
Syntax	`uns8` **`setFSR01(buffer0, buffer1)`**

Parameters	`buffer0`, `buffer1`:
	`_FSR_NINFO` Set `FSR` to `networkInfo`
	`_FSR_INFO` Set `FSR` to `bufferINFO`
	`_FSR_COM` Set `FSR` to `bufferCOM`
	`_FSR_AUX` Set `FSR` to `bufferAUX`
	`_FSR_RF` Set `FSR` to `bufferRF`
	`_FSR_ntwADDR` Set `FSR` to `ntwADDR`

Return value	64 Constant value to optimize possible subsequent work with buffers
Output values	• `FSR0` and `FSR1` address bytes[0] of specified OS buffers • `WREG = 64`
Preconditions	–
Remarks	See IQRF OS User's guide [1], chapter *Data memory (RAM)*.
Side effects	–
See also	`setFSR0`, `setFSR1`

Example	`setFSR0(_FSR_COM);  // FSR0 addresses bufferCOM[0]`
	`setFSR1(_FSR_RF);  // FSR1 addresses bufferRF[0]`
	`setINDF1(INDF0);  // bufferRF[0] = bufferCOM[0]`

5.5 Data blocks

5.5.1 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], chapter *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  "`
	`...`
	`eeWriteByte(__EEAPPINFO+29, '2');  // #01 changed to #02`
	`appInfo();  // "Application data, I'm user #02  " is read`

5.6 Networking

5.6.1 setFRCresponseTime

Macro	Specify the time needed to complete FRC responses in Nodes
Purpose	Synchronize waiting for FRC answers according to the slowest device
Syntax	`void setFRCresponseTime(ms)`

Parameters	`ms: _FRC_RESPONSE_TIME_40_MS  40 ms`
	`_FRC_RESPONSE_TIME_360_MS  360 ms`
	`_FRC_RESPONSE_TIME_680_MS  680 ms`
	`_FRC_RESPONSE_TIME_1320_MS  1 s 320 ms`
	`_FRC_RESPONSE_TIME_2600_MS  2 s 600 ms`
	`_FRC_RESPONSE_TIME_5160_MS  5 s 160 ms`
	`_FRC_RESPONSE_TIME_10280_MS  10 s 280 ms`
	`_FRC_RESPONSE_TIME_20520_MS  20 s 520 ms`

Return value	–
Output values	Response FRC time is selected from 8 possible values.
Preconditions	This time must be selected by the Coordinator before `sendFRC`.
Remarks	See `sendFRC`, *Preconditions*.
Side effects	–
See also	`sendFRC`, `getFRCresponseTime`
Example	See `sendFRC`, *Example 1*.

5.6.2 getFRCresponseTime

Macro	Get current value of FRC response time
Purpose	
Syntax	`uns8 getFRCresponseTime()`
Parameters	–

Return value	`0x00:  40 ms`
	`0x10:  360 ms`
	`0x20:  680 ms`
	`0x30:  1 s 320 ms`
	`0x40:  2 s 600 ms`
	`0x50:  5 s 150 ms`
	`0x60:  10 s 280 ms`
	`0x70:  20 s 520 ms`

Output values	–
Preconditions	–
Remarks	See `setFRCresponseTime` and `sendFRC`, *Preconditions*.
Side effects	–
See also	`sendFRC`, `setFRCresponseTime`
Example	`userValue = getFRCresponseTime();`

5.7 Compatibility

Macros in this chapter are intended for compatibility with older TR and/or OS versions.

Name	Interpretation	Remarks
`setTXpower(level)`	`setRFpower(level)`	OS function renamed in history
`prebondNode()`	`prebondNodeAtNode()`	Recent `prebondNode` has been replaced by `prebondNodeAtNode` and `prebondNodeAtCoordinator`. This substitution may not be exactly equivalent.
`reset()`	`softReset()`	Just an alias for MCU machine instruction `Reset` and CC5X native function
`breakpoint(wValue)`	`debugW(wValue)`	Renamed in history.

6 Documentation and information

1 IQRF OS User's guide
2 RAM map and EEPROM map, IQRF OS User's guide, Appendix 1
3 SPI specification
4 TR-72D datasheet or TR-76D datasheet
5 RF IC datasheet
6 PIC16LF1938 datasheet
7 Temperature sensor datasheet
8 IQRF IDE development environment
9 Examples (included in the StartUp Package)

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

7 Document revision

171109	`bondRequestAdvanced` and `removeBond` *Preconditions* slightly precised. Remarks in chapter *Encryption* slightly precised. (DC)TR-75D added to `moduleInfo()`. A bug in `_NTWF` flag fixed.
170821	Updated for IQRF OS v4.02D. TR-78D added to `moduleInfo` *Output values*. `encryptBufferRF` and `decryptBufferRF` *Remarks* slightly precised. `getRSSI` *Preconditions* and *Example* slightly precised. A bug in `responseFRC()` *Example* fixed. Bugs in the `setFRCresponseTime` macro name and `_FRC_RESPONSE_TIME…` constants names fixed. Return and output values added at macros `setFSR0`, `setFSR1` and `setFSR01`. A bug in `checkRF(x)` parameter range fixed. Return values added to *Table of macros*.
170810	Updated for IQRF OS v4.01D. Removed due to a serious bug in OS v4.01D.
170322	Minor bugs regarding removed function `setUserAddress` fixed.
170314	First release for IQRF OS v4.00.

8 Index

9 Sales and Service

9.1 Corporate office

IQRF Tech s.r.o., Prumyslova 1275, 506 01 Jicin, Czech Republic, EU
Tel: +420 493 538 125, Fax: +420 493 538 126, www.iqrf.tech
E-mail (commercial matters): sales@iqrf.org

9.2 Technology and development

www.iqrf.org
E-mail (technical matters): support@iqrf.org

9.3 Partners and distribution

www.iqrf.org/partners

9.4 Quality management

ISO 9001 : 2009 certified

9.5 Trademarks

The IQRF name and logo are registered trademarks of IQRF Tech s.r.o.

PIC, SPI, Microchip and all other trademarks mentioned herein are property of their respective owners.

9.6 Legal

On-line support: support@iqrf.org
