

# **IQRF OS**

## **Operating System**

**Version 3.04D  
for TR-5xD**

## **Reference Guide**



## 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 RAM map [2]. Individual functions have up to 3 parameters. Several functions use some of these params and W (PIC accumulator) to return output values. Note that they are valid until another function using the same parameter or the debug function is called by the user. Additionally, some functions use some params as work variables that is why their previous content can be destroyed.

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

### Functions

<b>Control</b>		<b>4</b>
<code>calibrateTimer()</code>	Calibrate tick generator – obsolete	4
<code>iqrfSleep()</code>	Set the TR module in power saving mode (Sleep)	5
<code>setRFsleep()</code>	Set the RF IC in power saving mode (Sleep)	6
<code>setRFready()</code>	Set the RF IC in ready mode (wake-up from Sleep)	6
<code>debug()</code>	Enter the debug mode	7
<code>uns8 getSupplyVoltage()</code>	Get voltage level for battery check	8
<code>getTemperature()</code>	Temperature measurement	9
<b>Active waiting</b>		<b>10</b>
<code>waitMS(ms)</code>	Active waiting (time in ms)	10
<code>waitDelay(ticks)</code>	Active waiting (time in ticks)	10
<code>waitNewTick()</code>	Wait for a new tick	11
<b>Timing on background</b>		<b>11</b>
<code>startDelay(ticks)</code>	Start waiting (time in ticks)	13
<code>startLongDelay(ticks)</code>	Start long waiting (time in ticks)	13
<code>bit isDelay()</code>	Still waiting	14
<code>startCapture()</code>	Resets counter of ticks	11
<code>captureTicks()</code>	Get number of ticks counted	12
<b>LED indication</b>		<b>15</b>
<code>setOnPulsingLED(ticks)</code>	LEDR and LEDG On times setting (for blinking)	15
<code>setOffPulsingLED(ticks)</code>	LEDR and LEDG Off times setting (for blinking)	15
<code>pulsingLEDR()</code>	Red LED activation (blinking on background)	16
<code>pulseLEDR()</code>	Single red LED pulse (one flash on background)	16
<code>stopLEDR()</code>	Red LED off, blinking stopped	17
<code>pulsingLEDG()</code>	Green LED activation (blinking on background)	17
<code>pulseLEDG()</code>	Single green LED pulse (one flash on background)	18
<code>stopLEDG()</code>	Green LED off, blinking stopped	18
<b>MCU EEPROM</b>		<b>19</b>
<code>uns8 eeReadByte(addr)</code>	Read one byte	19
<code>eeReadData(addr, length)</code>	Read a block	19
<code>eeWriteByte(addr, data)</code>	Write one byte	20
<code>eeWriteData(addr, length)</code>	Write a block	20
<b>Serial EEPROM</b>		<b>21</b>
<code>eeeReadData(addr)</code>	Read a 16 B block from serial EEPROM to <code>bufferINFO</code>	21
<code>eeeWriteData(addr)</code>	Write a 16 B block from <code>bufferINFO</code> to EEPROM	21
<b>RAM</b>		<b>22</b>
<code>uns8 readFromRAM(addr)</code>	Read one byte	22
<code>writeToRAM(addr, data)</code>	Write one byte	23
<code>void setINDF0(value)</code>	Indirect write via virtual <code>INDF0</code> register	24
<code>void setINDF1(value)</code>	Indirect write via virtual <code>INDF1</code> register	24
<code>uns8 getINDF0()</code>	Indirect read via virtual <code>INDF0</code> register	25
<code>uns8 getINDF1()</code>	Indirect read via virtual <code>INDF1</code> register	26
<b>Buffers</b>		<b>27</b>
<code>clearBufferINFO()</code>	<code>bufferINFO</code> clearing	31
<code>clearBufferRF()</code>	<code>bufferRF</code> clearing	31
<code>copyBufferINFO2COM()</code>	Copy <code>bufferINFO</code> to <code>bufferCOM</code>	27
<code>copyBufferINFO2RF()</code>	Copy <code>bufferINFO</code> to <code>bufferRF</code>	27
<code>copyBufferRF2COM()</code>	Copy <code>bufferRF</code> to <code>bufferCOM</code>	28
<code>copyBufferRF2INFO()</code>	Copy <code>bufferRF</code> to <code>bufferINFO</code>	28
<code>copyBufferCOM2RF()</code>	Copy <code>bufferCOM</code> to <code>bufferRF</code>	29

<code>copyBufferCOM2INFO ()</code>	Copy <code>bufferCOM</code> to <code>bufferINFO</code>	29
<code>bit compareBufferINFO2RF (length)</code>	Comparison of <code>bufferINFO</code> and <code>bufferRF</code>	30
<code>void swapBufferINFO ()</code>	Swap <code>bufferINFO</code> and <code>bufferAUX</code>	30
<b>Data blocks</b>		<b>32</b>
<code>copyMemoryBlock (uns16 from, uns16 to, uns8 length)</code>	Copy any data block to any position	32
<code>moduleInfo ()</code>	Get info about transceiver module and OS	33
<code>appInfo ()</code>	Copy info about application from EEPROM to <code>bufferINFO</code>	34
<b>SPI</b>		<b>35</b>
<code>enableSPI ()</code>	SPI communication line activation	35
<code>disableSPI ()</code>	SPI communication line deactivation	35
<code>startSPI (length)</code>	SPI packet transmission	36
<code>stopSPI ()</code>	SPI stopping	37
<code>restartSPI ()</code>	SPI continuing	37
<code>bit getStatusSPI ()</code>	SPI status, update SPI flags	38
<b>RF</b>		<b>39</b>
<code>setTXpower (level)</code>	RF power setting (7 levels)	39
<code>setRFspeed (speed)</code>	Select RF bit rate	39
<code>setRFband (band)</code>	Select RF band – obsolete	40
<code>setRFchannel (channel)</code>	Select RF channel	40
<code>setRFmode (mode)</code>	Select RF power management mode	41
<code>checkRF (level)</code>	Detect incoming RF signal	43
<code>getRSSI ()</code>	Get RSSI value of incoming RF signal	45
<code>RFTXpacket ()</code>	Send a packet from <code>bufferRF</code> via RF	48
<code>bit RFRXpacket ()</code>	Receive a packet via RF to <code>bufferRF</code>	48
<b>Networking</b>		<b>50</b>
<code>setCoordinatorMode ()</code>	Device is the Coordinator	50
<code>setNodeMode ()</code>	Device is a Node	50
<code>setNonetMode ()</code>	Networking disabled	51
<code>setNetworkFilteringOn ()</code>	Packets accepted from current network only	52
<code>setNetworkFilteringOff ()</code>	Packets accepted from both networks	52
<code>setUserAddress (uns16: address)</code>	Assign a user address to a Node	53
<code>uns8 getNetworkParams ()</code>	Get information about the network	54
<b>Routing</b>		<b>55</b>
<code>setRoutingOn ()</code>	Outgoing packets routed via other devices on background	55
<code>setRoutingOff ()</code>	No routing for outgoing packets	55
<code>uns8 discovery (zones)</code>	Discover Nodes for routing	56
<code>answerSystemPacket ()</code>	Enable response to Coordinator for Discovery	57
<code>bit isDiscoveredNode (N)</code>	Check for being discovered	58
<code>bit wasRouted ()</code>	Indicate incoming packet routing	58
<code>optimizeHops (x)</code>	Optimize number of hops for given Node	59
<b>Bonding - Node</b>		<b>60</b>
<code>bit bondRequestAdvanced ()</code>	Request for bonding (local or remote)	60
<code>bit bondRequest (-)</code>	Request for bonding – obsolete	60
<code>prebondNode ()</code>	Preparing Node for remote bonding	62
<code>bit amIBonded ()</code>	Is the Node bonded?	63
<code>removeBond ()</code>	Unbonding	63
<b>Bonding - Coordinator</b>		<b>64</b>
<code>bit bondNewNode (address)</code>	Local bonding a Node	64
<code>nodeAuthorization ()</code>	Remote bonding of prebonded Node	65
<code>bit isBondedNode (N)</code>	Is the Node bonded?	66
<code>removeBondedNode (N)</code>	Unbonding a Node	67
<code>bit rebondNode (N)</code>	Rebonding a Node	67
<code>clearAllBonds ()</code>	Clearing of all bonds	68
<b>RFPGM</b>		<b>69</b>
<code>enableRFPGM ()</code>	Set to switch to RFPGM mode after reset	69
<code>disableRFPGM ()</code>	Set not to switch to RFPGM mode after reset	69
<code>runRFPGM ()</code>	Switch to RFPGM mode	70
<code>setupRFPGM (x)</code>	Setup RFPGM parameters	71

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

**OS functions****Control****calibrateTimer**

<b>Function</b>	Calibrate tick generator
<b>Purpose</b>	–
<b>Syntax</b>	<code>void calibrateTimer()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Do not use this function. It is not necessary to calibrate IQMESH timing from OS v3.01D.
<b>Remarks</b>	–
<b>Side effects</b>	–
<b>See also</b>	–
<b>Example</b>	–

Obsolete

### iqrfsleep

<b>Function</b>	Setting the TR module in power saving mode (Sleep)
<b>Purpose</b>	Easy and efficient power management. This function, once called, puts the module into the Sleep mode. Wake-up can be caused by power off/on, watchdog timeout or on the C5 (for TR modules in SIM format, e.g. TR-52D) or Q12 (for TR-54D) pin change.
<b>Syntax</b>	<code>void iqrfsleep ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• This functions operates like the PIC machine instruction Sleep. Additionally, OS suspends all HW resources that are under its control (RF circuitry, timers, internal PIC pins, LEDs etc.). The user should do the same for resources used by the application before entering the Sleep mode to achieve minimal power consumption.</li> <li>• For wake-up on pin change the required sequence should be executed. Wake-up on pin change is default disabled.</li> <li>• This function is not time-efficient for subsequent short sleep periods, especially if RF IC is off. For faster operation in such cases use <code>sleep()</code> instead but you should ensure minimal consumption by user program. See Example 3.</li> </ul>
<b>Remarks</b>	All features are under user's control. RBIF flag is not cleared to allow to distinguish wake-up type. See example E01-TX [10].
<b>Side effects</b>	Global interrupt enable (GIE) is controlled by OS again after wake-up.
<b>See also</b>	<code>setRFsleep</code>
<b>Example 1</b>	<pre> // Minimize consumption (depends on resources used by the user) Motor = 0;           // Stop the motor ADON = 0;           // Disable A/D converter SWDTEN = 0;         // Disable watchdog iqrfsleep();        // Put the module into Sleep mode </pre>
<b>Example 2</b>	<pre> // Wake-up on pin change. See example E01-TX and IQRF-macros.h header file. GIE = 0;           // Disable all interrupts writeToRAM(&amp;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 IOCFIE = 1;        // Interrupt on change enabled SWDTEN = 0;        // Watchdog disabled iqrfsleep();       // Sleep writeToRAM(&amp;IOCBF, IOCBF &amp; 0xEF); // Clear interrupt on change flag. // Instead of IOCBF.4=0; // Bit IOCBF.4 cannot be accessed // directly due to OS restriction. if (buttonPressed) // If button is pressed { ... }           // ... </pre>
<b>Example 3</b>	<pre> 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 </pre>

### setRFsleep

<b>Function</b>	Setting RF circuitry in power saving mode (Sleep)
<b>Purpose</b>	To put all RF circuitry in Sleep mode. Easy and efficient power management.
<b>Syntax</b>	<code>void setRFsleep ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	<ul style="list-style-type: none"> <li>• RF IC is set off.</li> <li>• OS system clock (ticks) are derived from MCU internal RC oscillator instead of precise RF IC crystal.</li> </ul>
<b>Preconditions</b>	–
<b>Remarks</b>	0.6 mA typ. is saved. RF response is prolonged for 2 ms typ., 7 ms max. due to wake-up. Wake-up can be caused by <code>setRFready</code> , <code>RFTXpacket</code> , <code>RFRXpacket</code> , <code>checkRF</code> or <code>getSupplyVoltage</code> .
<b>Side effects</b>	–
<b>See also</b>	<code>setRFready</code> , <code>iqrfSleep</code> , <code>getSupplyVoltage</code> , <code>checkRF</code> , <code>RFTXpacket</code> , <code>RFRXpacket</code>
<b>Example</b>	<code>setRFsleep(); // Put the RF circuitry in Sleep mode</code>

### setRFready

<b>Function</b>	Wake RF circuitry up
<b>Purpose</b>	To wake RF circuitry up in advance for faster response, easy and efficient power management and precise ticks.
<b>Syntax</b>	<code>void setRFready ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	<ul style="list-style-type: none"> <li>• RF IC is set on (the RF ready mode) but RX chain still stays off (unlike the RX mode). See IQRF User's guide, RF IC modes.</li> <li>• RF IC crystal oscillator starts up.</li> <li>• OS system clock (tick) is based on precise RF IC crystal oscillator instead of MCU internal RC one.</li> </ul>
<b>Preconditions</b>	–
<b>Remarks</b>	Takes ~7 ms. After this time the RX chain can be set on faster which enables faster <code>checkRF(x)</code> , <code>RFRXpacket()</code> or <code>RFTXpacket()</code> .
<b>Side effects</b>	–
<b>See also</b>	<code>setRFsleep</code> , <code>iqrfSleep</code> , <code>getSupplyVoltage</code> , <code>checkRF</code> , <code>RFTXpacket</code> , <code>RFRXpacket</code>
<b>Example</b>	<pre>setRFready(); // Wake the RF circuitry up from RF sleep in advance ... RFTXpacket(); // for immediate response</pre>

## debug

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

### getSupplyVoltage

<b>Function</b>	Power supply measurement (up to 3.8 V)
<b>Purpose</b>	Battery check (for discharge-sensitive batteries)
<b>Syntax</b>	<code>uns8 getSupplyVoltage ()</code>
<b>Parameters</b>	–
<b>Return value</b>	level = 1, 2, ...15                      Voltage > 2.25 V + level × 0.1 V
<b>Output values</b>	–
<b>Preconditions</b>	–
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Internal power supply voltage is checked.</li> <li>• 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.</li> <li>• To evaluate the battery, take into consideration your battery type and power supply circuitry with respect to diodes and other possible voltage drops.</li> <li>• The detector circuit has a built-in 50 mV hysteresis.</li> </ul>
<b>Side effects</b>	The RF circuitry wakes up (in case of sleeping).
<b>See also</b>	<code>setRFsleep</code>
<b>Example</b>	<pre> if (getSupplyVoltage () &lt; 7)     ...                // Low battery else     ...                // Voltage &gt; 2.95V </pre>



### getTemperature

<b>Function</b>	Read temperature from on-board sensor																														
<b>Purpose</b>	Temperature measurement																														
<b>Syntax</b>	uns8 <b>getTemperature</b> ()																														
<b>Parameters</b>	–																														
<b>Return value</b>	<ul style="list-style-type: none"> <li>• Temperature in °C, integer part, not rounded</li> <li>• Negative temperatures are in two's complement format (e.g. 0xFFB means -5 °C)</li> <li>• 0x80 (-128 °C). For TR modules without temperature sensor, i.e. without the "T" postfix, e.g. TR-52D.</li> </ul>																														
<b>Output values</b>	<p>param3: complete 12 b output value of the sensor in 0.0625 °C units. Thus, lower 4 b represent the fractional part and upper 8 b represent the integer part of temperature. Negative temperatures are in two's complement format. See datasheet of the sensor.</p> <p>Examples:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Temperature</th> <th>Return value</th> <th>param3</th> <th>Temperature</th> <th>Return value</th> <th>param3</th> </tr> </thead> <tbody> <tr> <td>50 °C</td> <td>0x32</td> <td>0x320</td> <td>0 °C</td> <td>0x00</td> <td>0x000</td> </tr> <tr> <td>5 °C</td> <td>0x05</td> <td>0x050</td> <td>-0.5 °C</td> <td>0xFF</td> <td>0xFF8</td> </tr> <tr> <td>5.5 °C</td> <td>0x05</td> <td>0x058</td> <td>-1 °C</td> <td>0xFF</td> <td>0xFF0</td> </tr> <tr> <td>0.75 °C</td> <td>0x00</td> <td>0x00C</td> <td>-8 .25 °C</td> <td>0xF7</td> <td>0xF7C</td> </tr> </tbody> </table>	Temperature	Return value	param3	Temperature	Return value	param3	50 °C	0x32	0x320	0 °C	0x00	0x000	5 °C	0x05	0x050	-0.5 °C	0xFF	0xFF8	5.5 °C	0x05	0x058	-1 °C	0xFF	0xFF0	0.75 °C	0x00	0x00C	-8 .25 °C	0xF7	0xF7C
Temperature	Return value	param3	Temperature	Return value	param3																										
50 °C	0x32	0x320	0 °C	0x00	0x000																										
5 °C	0x05	0x050	-0.5 °C	0xFF	0xFF8																										
5.5 °C	0x05	0x058	-1 °C	0xFF	0xFF0																										
0.75 °C	0x00	0x00C	-8 .25 °C	0xF7	0xF7C																										
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• 100 ms delay is required after wake up from sleep</li> </ul>																														
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• TMP112 (TI) temperature sensor is used</li> <li>• Resolution 0.0625 °C, accuracy: 0.5 °C</li> <li>• Takes ~1.7 ms</li> <li>• See example E08–TEMPERATURE [10]</li> </ul>																														
<b>Side effects</b>	–																														
<b>See also</b>	–																														
<b>Example 1</b>	<pre> // For positive temperatures only uns8  tempInt;                // Temperature, integer part uns8  tempFract;             // Temperature, fractional part tempInt = getTemperature(); tempFract = param3.low8 &amp; 0x0F // Temperature == tempInt + tempFract/16 // Temperature == param3 * 0.0625 in °C </pre>																														
<b>Example 2</b>	<pre> // Either positive or negative temperatures, fractional part ignored T = getTemperature(); // Integer part of temperature if (T &gt;= 0x80) {     sign = "-"; // Negative     T = (T ^ 0xFF) + 1; // Get absolute value in °C } else     sign = "+"; // Positive </pre>																														
<b>Example 3</b>	<pre> // Either positive or negative temperatures, with fractional part if (getTemperature() &gt;= 0x80) {     sign = "-"; // Negative     T = (param3 ^ 0xFFF) + 1; // Get absolute value, in 0.0625°C units } else     sign = "+"; // Positive </pre>																														
<b>Example 4</b>	<pre> // Temperature measurement after wake-up from sleep iqrfsleep(); waitDelay(10); // 100 ms delay required T = getTemperature(); </pre>																														

## Active waiting

### waitMS

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

### waitDelay

<b>Function</b>	Wait specified number of ticks
<b>Purpose</b>	Time delay generation
<b>Syntax</b>	<code>void waitDelay (ticks)</code>
<b>Parameters</b>	ticks – time to wait in 10 ms periods (1 - 255)
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	This function can be combined with <code>waitMS</code> .
<b>Remarks</b>	This is the active waiting (on OS foreground). No other operation runs on OS foreground during waiting.
<b>Side effects</b>	<ul style="list-style-type: none"> <li>• This function must not be combined with <code>startDelay</code> and <code>startLongDelay</code>.</li> <li>• Internal ticks are based on internal RC oscillator. Thus, the delay can vary with temperature etc. See respective PIC datasheet [8].</li> <li>• Delay in first tick can vary from 0 ms to 10 ms. If complete 10 ms is needed also in the first tick, use <code>waitNewTick</code> firstly.</li> <li>• For short time delays <code>waitMS</code> is more precise.</li> </ul>
<b>See also</b>	<code>waitMS</code> , <code>startDelay</code> , <code>startLongDelay</code>
<b>Example 1</b>	<pre>    // LED on for 0.5 s _LED = 1; waitDelay(50);    // Delay 500 ms. Program stays here for 500 ms _LED = 0;        // and continues here just after the period elapsed.</pre>

### waitNewTick

<b>Function</b>	Wait for a new tick
<b>Purpose</b>	Timing synchronization of user operations
<b>Syntax</b>	<code>void waitNewTick ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	–
<b>Remarks</b>	Active waiting (on OS foreground) until a new tick starts. No other operation runs on OS foreground during this waiting.
<b>Side effects</b>	–
<b>See also</b>	<code>waitMS</code> , <code>waitDelay</code>
<b>Example</b>	<pre>waitNewTick();    // To generate a pulse as precise as possible IO1 = 1; waitDelay(1);    // 10 ms IO1 = 0;</pre>

### Timing on background

#### startCapture

<b>Function</b>	Reset and start the Capture timer
<b>Purpose</b>	Initialization of time measurement or delay generation
<b>Syntax</b>	<code>void startCapture ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	This function can be combined with <code>waitMS</code> .
<b>Remarks</b>	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.
<b>Side effects</b>	Functionality is affected by <code>bondRequest</code> , <code>bondNewNode</code> , <code>RFRXpacket</code> and <code>RFTXpacket</code> .
<b>See also</b>	<code>captureTicks</code>
<b>Example</b>	See <code>captureTicks</code>

### captureTicks

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

### startDelay

<b>Function</b>	Preset and start the Delay timer
<b>Purpose</b>	Initialization of time measurement or delay generation
<b>Syntax</b>	void <b>startDelay</b> ( <b>ticks</b> )
<b>Parameters</b>	uns8 ticks: number of ticks (10 ms system intervals) to be measured (1-255)
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	This function can be combined with <code>waitMS</code> and <code>startLongDelay</code> .
<b>Remarks</b>	The Delay timer measures specified time period on OS background. The result is available via the <code>isDelay</code> function.
<b>Side effects</b>	<ul style="list-style-type: none"> <li>• This function does not work properly if the <code>waitDelay</code> or <code>startLongDelay</code> functions are active.</li> <li>• Functionality is affected by <code>bondRequest</code>, <code>bondNewNode</code>, <code>RFRXpacket</code> and <code>RFTXpacket</code>.</li> <li>• Delay in first tick can vary from 0 ms to 10 ms. If complete 10 ms is needed also in the first tick, use <code>waitNewTick</code> firstly.</li> <li>• Internal ticks are based on internal RC oscillator if RF IC is sleeping. Thus, the delay can vary with temperature etc. in this case. See respective PIC datasheet [8].</li> </ul>
<b>See also</b>	<code>isDelay</code> , <code>startLongDelay</code> , <code>waitDelay</code>
<b>Example</b>	See <code>isDelay</code>

### startLongDelay

<b>Function</b>	Preset and start the LongDelay timer
<b>Purpose</b>	Initialization of time measurement or delay generation
<b>Syntax</b>	void <b>startLongDelay</b> ( <b>ticks</b> )
<b>Parameters</b>	uns16 ticks: number of ticks (10 ms system intervals) to be measured (1-65535)
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	This function can be combined with <code>waitMS</code> and <code>startDelay</code> .
<b>Remarks</b>	The Delay timer measures specified time period on OS background. The result is available via the <code>isDelay</code> function.
<b>Side effects</b>	<ul style="list-style-type: none"> <li>• This function does not work properly if the <code>waitDelay</code> or <code>startLongDelay</code> functions are active.</li> <li>• Functionality is affected by <code>bondRequest</code>, <code>bondNewNode</code>, <code>RFRXpacket</code> and <code>RFTXpacket</code>.</li> <li>• Delay in first tick can vary from 0 ms to 10 ms. If complete 10 ms is needed also in the first tick, use <code>waitNewTick</code> firstly.</li> <li>• Internal ticks are based on internal RC oscillator if RF IC is sleeping. Thus, the delay can vary with temperature etc. in this case. See respective PIC datasheet [8].</li> </ul>
<b>See also</b>	<code>isDelay</code> , <code>startDelay</code> , <code>waitDelay</code>
<b>Example</b>	See <code>isDelay</code>

## isDelay

<b>Function</b>	Information whether specified delay is still in progress
<b>Purpose</b>	Time measurement or delay generation
<b>Syntax</b>	bit <code>isDelay()</code>
<b>Parameters</b>	–
<b>Return value</b>	<ul style="list-style-type: none"> <li>• 1: still in progress</li> <li>• 0: elapsed</li> </ul>
<b>Output values</b>	–
<b>Preconditions</b>	<code>startDelay</code> or <code>startLongDelay</code> should be used before.
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• The (Long)Delay timer measures specified time period. The result is available via the <code>isDelay</code> function.</li> <li>• Tip: the <code>clrwdt</code> instruction should be used to avoid unintentional watchdog reset during the delay.</li> <li>• See example E05-DELAYS [10].</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>startDelay</code> , <code>startLongDelay</code>
<b>Example 1</b>	<pre> // LED on for 1 s _LED = 1; startDelay(100);           // Start 1 sec delay counting on OS background while (isDelay())         // Wait until the delay is over {     clrwdt();              // Any useful operation on OS foreground can be     ...                    // performed during waiting } _LED = 0;                  // Continue here after 1 sec </pre>
<b>Example 2</b>	<pre> // LED on for 10 s _LED = 1; startLongDelay(1000);     // Start 10 sec delay counting on OS background while (isDelay())        // Wait until the delay is over {     clrwdt();              // Any useful operation on OS foreground can be     ...                    // performed during waiting } _LED = 0;                // Continue here after 10 sec </pre>

**LED indication**
**setOnPulsingLED**

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

**setOffPulsingLED**

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

### pulsingLEDR

<b>Function</b>	Red LED blinking
<b>Purpose</b>	Continuous red LED blinking on OS background
<b>Syntax</b>	<code>void pulsingLEDR()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• Blinking times should be defined in advance by <code>setOnPulsingLED</code> and <code>setOffPulsingLED</code>.</li> <li>• The appropriate PIC pin is configured as an output automatically.</li> <li>• Do not combine this function with direct access to LED pin (see <code>pulseLEDR</code> <i>Remarks</i>). If omitted, the pin state can be modified in background.</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Blinking continues until it is stopped by the user (e.g. by <code>stopLEDR</code>).</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>setOnPulsingLED</code> , <code>setOffPulsingLED</code> , <code>stopLEDR</code> , <code>pulseLEDR</code>
<b>Example 1</b>	<code>pulsingLEDR(); // continuous blinking on OS background</code>
<b>Example 2</b>	<pre>// Blinking for 2 s pulsingLEDR(); // blinking for 2 s on OS background waitDelay(200); // 2 s delay generated on foreground stopLEDR(); // Stop blinking</pre>

### pulseLEDR

<b>Function</b>	Single red LED flash
<b>Purpose</b>	Red LED flash on OS background
<b>Syntax</b>	<code>void pulseLEDR()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• Flash time should be defined in advance by <code>setOnPulsingLED</code>.</li> <li>• The appropriate PIC pin is configured as an output automatically.</li> <li>• Do not combine this function with direct access to LED pin (see <i>Remarks</i>). If omitted, the pin state can be modified in background.</li> </ul>
<b>Remarks</b>	The on-board LEDs can also be directly controlled on OS foreground using C commands for manipulating the <code>_LEDR</code> output (the pin the red LED is connected to) and corresponding control bit ( <code>TRISx.x</code> - see <code>IQRF-memory.h</code> header file), e.g. <code>_LEDR = 1</code> .
<b>Side effects</b>	–
<b>See also</b>	<code>setOnPulsingLEDR</code> , <code>pulsingLEDR</code> , <code>stopLEDR</code>
<b>Example</b>	<pre>setOnPulsingLEDR(10); // 100 ms On pulseLEDR(); // Single red LED flash for 100 ms on OS background ... // Program continues immediately, // not waiting until the delay expires. // LED will be switched off after 100 ms automatically</pre>



### stopLEDR

<b>Function</b>	Red LED off, blinking stopped
<b>Purpose</b>	Stops the red LED activity on OS background
<b>Syntax</b>	<code>void stopLEDR ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Do not combine this function with direct access to LED pin (see <code>pulseLEDR</code> <i>Remarks</i> ). If omitted, the pin state can be modified in background.
<b>Remarks</b>	–
<b>Side effects</b>	–
<b>See also</b>	<code>pulsingLEDR</code> , <code>pulseLEDR</code>
<b>Example 1</b>	<pre>pulsingLEDR ();           // Start blinking on OS background ...                       // Blinking continues during any operation stopLEDR ();             // Stop blinking</pre>
<b>Example 2</b>	<pre>pulseLEDR ();           // Red LED On on OS background ...                       // continuously lighting during any operation                         // until specified time expired stopLEDR ();           // or LED is switched Off by this command</pre>
<b>Example 3</b>	<pre>_LEDR = 1;             // LEDR on ... stopLEDR ();           // LEDR off</pre>

### pulsingLEDG

<b>Function</b>	Green LED blinking
<b>Purpose</b>	Continuous green LED blinking on OS background
<b>Syntax</b>	<code>void pulsingLEDG ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• Blinking times should be defined in advance by <code>setOnPulsingLED</code> and <code>setOffPulsingLED</code>.</li> <li>• The appropriate PIC pin is configured as an output automatically.</li> <li>• Do not combine this function with direct access to LED pin (see <code>pulseLEDG</code> <i>Remarks</i>). If omitted, the pin state can be modified in background.</li> </ul>
<b>Remarks</b>	Blinking continues until it is stopped by the user (e.g. by <code>stopLEDG</code> ).
<b>Side effects</b>	–
<b>See also</b>	<code>setOnPulsingLED</code> , <code>setOffPulsingLED</code> , <code>stopLEDG</code> , <code>pulseLEDG</code>
<b>Example 1</b>	<pre>pulsingLEDG ();         // continuous blinking on OS background</pre>
<b>Example 2</b>	<pre>// Blinking for 2 s pulsingLEDG ();         // blinking for 2 s on OS background waitDelay(200);         // 2 s delay generated on foreground stopLEDG ();           // Stop blinking</pre>

### pulseLEDG

<b>Function</b>	Single green LED flash
<b>Purpose</b>	Green LED flash on OS background
<b>Syntax</b>	<code>void pulseLEDG ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>Flash time should be defined in advance by <code>setOnPulsingLED</code>.</li> <li>The appropriate PIC pin is configured as an output automatically.</li> <li>Do not combine this function with direct access to LED pin (see <i>Remarks</i>). If omitted, the pin state can be modified in background.</li> </ul>
<b>Remarks</b>	The on-board LEDs can also be directly controlled on OS foreground using C commands for manipulating the <code>_LEDG</code> output (the pin the green LED is connected to) and corresponding control bit ( <code>TRISx.x</code> - see <code>IQRF-memory.h</code> header file), e.g. <code>_LEDG = 1</code> .
<b>Side effects</b>	–
<b>See also</b>	<code>setOnPulsingLEDG</code> , <code>pulsingLEDG</code> , <code>stopLEDG</code>
<b>Example</b>	<pre>setOnPulsingLEDG(10); // 100 ms On pulseLEDG();          // Single green LED flash for 100 ms on OS background ...                  // Program continues immediately,                     // not waiting until the delay expires.                     // LED will be switched off after 100 ms automatically</pre>

### stopLEDG

<b>Function</b>	Green LED off, blinking stopped
<b>Purpose</b>	Stops the green LED activity on OS background
<b>Syntax</b>	<code>void stopLEDG ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Do not combine this function with direct access to LED pin (see <code>pulseLEDG</code> <i>Remarks</i> ). If omitted, the pin state can be modified in background.
<b>Remarks</b>	–
<b>Side effects</b>	<ul style="list-style-type: none"> <li>The appropriate PIC pin is not restored to the state before <code>pulsingLEDG/pulseLEDG</code> (<code>TRISx.x == 0, _LEDG == 0</code> after finishing on background).</li> <li>Possible user <code>LEDR</code> pin level (in <code>PORT</code> or <code>LATCH</code> register) changed in foreground can be overridden in background.</li> </ul>
<b>See also</b>	<code>pulsingLEDG</code> , <code>pulseLEDG</code>
<b>Example 1</b>	<pre>pulsingLEDG(); // Start blinking on OS background ...           // Blinking continues during any operation stopLEDG();   // Stop blinking</pre>
<b>Example 2</b>	<pre>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</pre>

## MCU EEPROM

### eeReadByte

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

### eeReadData

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

### eeWriteByte

<b>Function</b>	Write one byte to specified location in EEPROM
<b>Purpose</b>	Access to EEPROM
<b>Syntax</b>	<code>void eeWriteByte (addr, data)</code>
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• <code>uns8 addr</code>: address in EEPROM (0xA0 to 0xBF for Coordinator and 0 to 0xBF for other devices). See EEPROM map [2].</li> <li>• <code>uns8 data</code>: value to be written (0 to 255)</li> </ul>
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	–
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Direct user access to EEPROM (using registers <code>EECONx</code> etc.) is not allowed for security reasons, specialized OS functions are intended for this.</li> <li>• EEPROM area dedicated to OS (locations 0xC0 or higher) is not accessible. See example E04–EEPROM [10].</li> <li>• Any attempt to write to protected area above 0xBF leads to no operation.</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>eeReadByte</code> , <code>eeReadData</code> , <code>eeWriteData</code>
<b>Example 1</b>	<pre>eeWriteByte(191, 0x75) // store 0x75 to EEPROM to address 191 eeWriteByte(0x80, X)  // copy X to EEPROM to address 0x80</pre>
<b>Example 2</b>	<pre>// Illegal access: Avoid access to EEPROM locations 192 (0xC0) or higher eeWriteByte(198, 0x75); // Attempt to write to protected area - nothing is written.</pre>

### eeWriteData

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

## Serial EEPROM

### eeeReadData

<b>Function</b>	Read a 16 B data block from specified location in serial EEPROM to <code>bufferINFO</code>
<b>Purpose</b>	Block access to serial EEPROM
<b>Syntax</b>	<code>void eeeReadData (addr)</code>
<b>Parameters</b>	• <code>uns16 addr</code> : address in serial EEPROM (0 to 0x7FF).
<b>Return value</b>	–
<b>Output values</b>	<code>bufferINFO[0 to 15]</code>
<b>Preconditions</b>	Do not use for Coordinator in networks utilizing Discovery
<b>Remarks</b>	–
<b>Side effects</b>	–
<b>See also</b>	<code>eeeWriteData</code>
<b>Example</b>	<pre> eeeReadData(10); // copy 16B from serial EEPROM from address 10 to bufferINFO                 // bufferINFO[0] = serial EEPROM[10]                 //                ...                 // bufferINFO[15] = serial EEPROM[25] </pre>

### eeeWriteData

<b>Function</b>	Write a 16 B data block from <code>bufferINFO</code> to specified location in EEPROM
<b>Purpose</b>	Block access to serial EEPROM
<b>Syntax</b>	<code>void eeeWriteData (addr)</code>
<b>Parameters</b>	• <code>uns16 addr</code> : address in serial EEPROM (0 to 0x7FF).
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Do not use for Coordinator in networks utilizing Discovery
<b>Remarks</b>	–
<b>Side effects</b>	–
<b>See also</b>	<code>eeeReadData</code>
<b>Example</b>	<pre> eeeWriteData(5); // copy 16B from bufferINFO to serial EEPROM from address 5                 // EEPROM[5] = bufferINFO[0]                 //                ...                 // EEPROM[20] = bufferINFO[15] </pre>

**RAM**
**readFromRAM**

<b>Function</b>	Read one byte from specified location in RAM
<b>Purpose</b>	Indirect access to RAM registers
<b>Syntax</b>	<code>uns8 readFromRAM(addr)</code>
<b>Parameters</b>	<code>uns16 addr</code> : linear or traditional memory location address
<b>Return value</b>	Value read from specified location
<b>Output values</b>	–
<b>Preconditions</b>	–
<b>Remarks</b>	See example E06–RAM [10].
<b>Side effects</b>	FSR0 register is modified.
<b>See also</b>	<code>writeToRAM</code> , <code>copyMemoryBlock</code> , <code>getINDF0</code> , <code>getINDF1</code>
<b>Example</b>	<pre> for (i=0; i&lt;5; i++) {     A = readFromRAM(bufferRF + i);     ... } </pre>

**writeToRAM**

<b>Function</b>	Write one byte to specified location in RAM
<b>Purpose</b>	Indirect access to RAM registers
<b>Syntax</b>	<code>void writeToRAM(addr, value)</code>
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• <code>uns16 addr</code>: traditional or linear memory location address</li> <li>• <code>uns8 value</code>: value to be written</li> </ul>
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• Avoid writing to RAM areas dedicated to OS and to PIC special function registers otherwise OS can collapse. See RAM map [2].</li> </ul>
<b>Remarks</b>	RAM can be accessed either directly (using common C commands like <code>X = Y;</code> ) or indirectly. But indirect writing to the <code>INDFx</code> registers is not allowed. Due to security reasons all instructions writing to <code>INDFx</code> are removed during Upload. To avoid unintended behavior, all constructions writing to <code>INDFx</code> (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 <code>readFromRAM</code> , <code>writeToRAM</code> and <code>copyMemoryBlock</code> . See example E06–RAM [10].
<b>Side effects</b>	<code>FSR0</code> register is modified.
<b>See also</b>	<code>readFromRAM</code> , <code>copyMemoryBlock</code> , <code>setINDF0</code> , <code>setINDF1</code>
<b>Example 1</b>	<pre>// Not allowed. The compiler uses INDFx in such cases. for (i=0; i&lt;5; i++)     bufferRF[i] = i;</pre>
<b>Example 2</b>	<pre>// Correct for (i=0; i&lt;5; i++)     writeToRAM(bufferRF + i, i);</pre>

### setINDF0

<b>Function</b>	Write a value in the virtual INDF0 register
<b>Purpose</b>	Indirect write to RAM
<b>Syntax</b>	<code>void setINDF0 (value)</code>
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• <code>uns8 value</code>: value to be written</li> </ul>
<b>Return value</b>	–
<b>Output values</b>	Register addressed by the <code>FSR0H</code> and <code>FSR0L</code> is modified
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• <code>FSR0</code> (the <code>FSR0H</code> and <code>FSR0L</code> register pair) must be set before to define a destination. Traditional as well as linear address can be used.</li> <li>• Avoid writing to RAM areas dedicated to OS and to PIC special function registers otherwise OS can collapse. See RAM map [2].</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Simple writing to the <code>INDF0</code> virtual register is not allowed. Due to security reasons all instructions using <code>INDF0</code> are removed during Upload. To avoid unintended behavior all constructions using addressing via <code>INDF0</code> (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 <code>setINDF0</code>. See example E06–RAM [10].</li> <li>• Another possibility is using the <code>writeToRAM</code> function.</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>setINDF1</code> , <code>getINDF0</code> , <code>getINDF1</code> , <code>writeToRAM</code> , <code>copyMemoryBlock</code>
<b>Example</b>	See <code>getINDF0</code> (by analogy with <code>INDF1</code> )

### setINDF1

<b>Function</b>	Write a value in the virtual INDF1 register
<b>Purpose</b>	Indirect write to RAM
<b>Syntax</b>	<code>void setINDF1 (value)</code>
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• <code>uns8 value</code>: value to be written</li> </ul>
<b>Return value</b>	–
<b>Output values</b>	Register addressed by the <code>FSR1H</code> and <code>FSR1L</code> is modified
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• <code>FSR1</code> (the <code>FSR1H</code> and <code>FSR1L</code> register pair) must be set before to define a destination. Traditional as well as linear address can be used.</li> <li>• Avoid writing to RAM areas dedicated to OS and to PIC special function registers otherwise OS can collapse. See RAM map [2].</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Simple writing to the <code>INDF1</code> virtual register is not allowed. Due to security reasons all instructions using <code>INDF1</code> are removed during Upload. To avoid unintended behavior all constructions using addressing via <code>INDF1</code> (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 <code>setINDF1</code>. See example E06–RAM [10].</li> <li>• Another possibility is using the <code>writeToRAM</code> function.</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>setINDF0</code> , <code>getINDF0</code> , <code>getINDF1</code> , <code>writeToRAM</code> , <code>copyMemoryBlock</code>
<b>Example</b>	See <code>getINDF0</code>



## getINDF0

<b>Function</b>	Read a value in the INDF0 virtual register
<b>Purpose</b>	Indirect read from RAM
<b>Syntax</b>	uns8 <code>getINDF0 ()</code>
<b>Parameters</b>	–
<b>Return value</b>	Value of the register addressed by the FSR0H and FSR0L
<b>Output values</b>	–
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>FSR0 (the FSR0H and FSR0L register pair) must be set before to define a destination. Traditional as well as linear address can be used.</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>See example E06–RAM [10].</li> <li>Additionally, simple reading the INDF0 virtual register is allowed from IQRF OS v3.04. This is even less memory consuming than <code>getINDF0</code>. See Example 2 below.</li> <li>Another possibility is using the <code>readFromRAM</code> function.</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>setINDF0</code> , <code>setINDF1</code> , <code>getINDF1</code> , <code>readFromRAM</code> , <code>copyMemoryBlock</code>
<b>Example 1</b>	<pre> // Not allowed FSR0 = ADDR_FROM; FSR1 = ADDR_TO; for(i = 0; i &lt; 16; i++) {     X = INDF0;           // Allowed from OS v3.04     INDF1 = X;          // Illegal writing to INDF1 (see setINDF1)     FSR0++;     FSR1++; } </pre>
<b>Example 2</b>	<pre> // Allowed from OS v3.04 FSR0 = ADDR_FROM; FSR1 = ADDR_TO; for(i = 0; i &lt; 16; i++) {     X = INDF0;           // Allowed (and recommended) from OS v3.04     setINDF1(X);     FSR0++;     FSR1++; } </pre>
<b>Example 3</b>	<pre> // Also correct but more memory consuming FSR0 = ADDR_FROM; FSR1 = ADDR_TO; for(i = 0; i &lt; 16; i++) {     X = getINDF0();     setINDF1(X);     FSR0++;     FSR1++; } // Result is the same as copyMemoryBlock(ADDR_FROM, ADDR_TO, 16) </pre>

### getINDF1

<b>Function</b>	Read a value in the INDF1 virtual register
<b>Purpose</b>	Indirect read from RAM
<b>Syntax</b>	uns8 <b>getINDF1</b> ()
<b>Parameters</b>	–
<b>Return value</b>	Value of the register addressed by the FSR1H and FSR1L
<b>Output values</b>	–
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>FSR1 (the FSR1H and FSR1L register pair) must be set before to define a destination. Traditional as well as linear address can be used.</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>See example E06–RAM [10].</li> <li>Additionally, simple reading the INDF1 virtual register is allowed from IQRF OS v3.04. This is even less memory consuming than getINDF1. See getINDF0, Example 2 .</li> <li>Another possibility is using the readFromRAM function.</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	setINDF0, setINDF1, getINDF0, readFromRAM, copyMemoryBlock
<b>Example</b>	See getINDF0 (by analogy with INDF1)

## Buffers

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

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

### copyBufferINFO2COM

<b>Function</b>	Copy <code>bufferINFO</code> to <code>bufferCOM</code>
<b>Purpose</b>	Data transfer between buffers
<b>Syntax</b>	<code>void copyBufferINFO2COM()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 64 B is copied.</li> <li>• See example E06 - RAM [10].</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>clearBufferINFO</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
<b>Example 1</b>	<code>copyBufferINFO2COM();</code>
<b>Example 2</b>	<pre>memoryOffsetFrom = 0; // bufferINFO to be copied memoryOffsetTo = 10; // to bufferCOM starting from bufferCOM[10]. copyBufferINFO2COM; // Just first 54 B is copied (until bufferCOM full).</pre>

### copyBufferINFO2RF

<b>Function</b>	Copy <code>bufferINFO</code> to <code>bufferRF</code>
<b>Purpose</b>	Data transfer between buffers
<b>Syntax</b>	<code>void copyBufferINFO2RF()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 64 B is copied.</li> <li>• See example E06 - RAM [10].</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>clearBufferINFO</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
<b>Example</b>	<code>copyBufferINFO2RF();</code>

### copyBufferRF2COM

<b>Function</b>	Copy <code>bufferRF</code> to <code>bufferCOM</code>
<b>Purpose</b>	Data transfer between buffers
<b>Syntax</b>	<code>void copyBufferRF2COM()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 64 B is copied.</li> <li>• See example E06 - RAM [10].</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>clearBufferINFO</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
<b>Example</b>	<code>copyBufferRF2COM();</code>

### copyBufferRF2INFO

<b>Function</b>	Copy <code>bufferRF</code> to <code>bufferINFO</code>
<b>Purpose</b>	Data transfer between buffers
<b>Syntax</b>	<code>void copyBufferRF2INFO()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Copying is limited up to first 64 B of <code>bufferRF</code> only.</li> <li>• If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 64 B is copied.</li> <li>• See example E06 - RAM [10].</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>clearBufferINFO</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2COM</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
<b>Example</b>	<code>copyBufferRF2INFO();</code>

### copyBufferCOM2RF

<b>Function</b>	Copy <code>bufferCOM</code> to <code>bufferRF</code>
<b>Purpose</b>	Data transfer between buffers
<b>Syntax</b>	<code>void copyBufferCOM2RF ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 64 B is copied.</li> <li>• See example E06 - RAM [10].</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>clearBufferINFO</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
<b>Example</b>	<code>copyBufferCOM2RF ();</code>

### copyBufferCOM2INFO

<b>Function</b>	Copy <code>bufferCOM</code> to <code>bufferINFO</code>
<b>Purpose</b>	Data transfer between buffers
<b>Syntax</b>	<code>void copyBufferCOM2INFO ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	Offsets <code>memoryOffsetFrom</code> and <code>memoryOffsetTo</code> are applied (see above).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 64 B is copied.</li> <li>• See example E06 - RAM [10].</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>clearBufferINFO</code> , <code>copyBufferINFO2COM</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyMemoryBlock</code>
<b>Example</b>	<code>copyBufferCOM2INFO ();</code>

### compareBufferINFO2RF

<b>Function</b>	Compare <code>bufferINFO</code> and <code>bufferRF</code> with respect to specified length
<b>Purpose</b>	Buffer comparison
<b>Syntax</b>	bit <code>compareBufferINFO2RF (length)</code>
<b>Parameters</b>	uns8 <code>length</code> : number of bytes to be compared (1 to 64)
<b>Return value</b>	<ul style="list-style-type: none"> <li>• 1 – match</li> <li>• 0 – mismatch</li> </ul>
<b>Output values</b>	–
<b>Preconditions</b>	–
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Comparing is limited up to first 64 B of <code>bufferRF</code> only.</li> <li>• If <code>memoryOffsetFrom = 0</code> and <code>memoryOffsetTo = 0</code> complete 64 B is compared.</li> <li>• See example E06 - RAM [10].</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>clearBufferINFO</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2INFO</code> , <code>swapBufferINFO</code>
<b>Example</b>	<pre>if (!compareBufferINFO2RF(32))    // Compare 32 B     then Error = 1;                // Error if mismatch</pre>

### swapBufferINFO

<b>Function</b>	Swap <code>bufferINFO</code> and <code>bufferAUX</code>
<b>Purpose</b>	Temporary <code>bufferINFO</code> saving
<b>Syntax</b>	void <code>swapBufferINFO ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	Content of <code>bufferINFO</code> and <code>bufferAUX</code> (64 B) is swapped. See example E06 - RAM [10].
<b>Preconditions</b>	–
<b>Remarks</b>	–
<b>Side effects</b>	–
<b>See also</b>	<code>moduleInfo</code> , <code>appInfo</code>
<b>Example</b>	<pre>swapBufferInfo();           // Temporarily save bufferInfo to bufferAUX appInfo();                  // Get user data from EEPROM ... swapBufferInfo();           // and restore previous data in bufferInfo</pre>

### clearBufferINFO

<b>Function</b>	Clear <code>bufferINFO</code>
<b>Purpose</b>	<code>bufferINFO</code> clearing
<b>Syntax</b>	<code>void clearBufferINFO ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	–
<b>Remarks</b>	Complete <code>bufferINFO</code> (64 B) is cleared (filled with zeros). See example E06 - RAM [10].
<b>Side effects</b>	–
<b>See also</b>	<code>copyBufferINFO2COM</code> , <code>copyBufferINFO2RF</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2INFO</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code> , <code>swapBufferINFO</code>
<b>Example</b>	<code>clearBufferINFO ();</code>

### clearBufferRF

<b>Function</b>	Clear <code>bufferRF</code>
<b>Purpose</b>	<code>bufferRF</code> clearing
<b>Syntax</b>	<code>void clearBufferRF ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	–
<b>Remarks</b>	Complete <code>bufferRF</code> (128 B) is cleared (filled with zeros). See example E06 - RAM [10].
<b>Side effects</b>	–
<b>See also</b>	<code>copyBufferRF2COM</code> , <code>copyBufferRF2INFO</code> , <code>copyBufferCOM2RF</code> , <code>copyBufferINFO2RF</code> , <code>compareBufferINFO2RF</code> , <code>copyMemoryBlock</code>
<b>Example</b>	<code>clearBufferRF ();</code>

**Data blocks**
**copyMemoryBlock**

<b>Function</b>	Copy specified RAM block to specified location
<b>Purpose</b>	Copy memory block within RAM
<b>Syntax</b>	<code>void copyMemoryBlock (from, to, length)</code>
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• <code>uns16 from</code>: starting address of the block to be copied</li> <li>• <code>uns16 to</code>: destination address</li> <li>• <code>uns8 length</code>: block length in bytes</li> </ul>
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• Either traditional or linear addresses can be used.</li> <li>• Upward overlapping the source and the destination RAM blocks being copied is not allowed.</li> <li>• Avoid writing to RAM areas dedicated to OS otherwise OS can collapse. See the RAM map [2].</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• See RAM map [2] and example E06 - RAM [10].</li> </ul>
<b>Side effects</b>	FSR0 and FSR1 registers are modified.
<b>See also</b>	<code>writeToRAM</code> , <code>readFromRAM</code> , <code>setINDF0</code> , <code>getINDF0</code> , <code>setINDF1</code> , <code>getINDF0</code>
<b>Example 1</b>	<code>copyMemoryBlock(0x2390, 0x23C0, 10); // copy 10 B block from 0x2390 to 0x23C0</code>
<b>Example 2</b>	<code>copyMemoryBlock(bufferRF+10, bufferCOM+1, 8); // 8 bytes copied: // bufferCOM[1] = bufferRF[10] ... bufferCOM[8] = bufferRF[17]</code>
<b>Example 3</b>	<code>copyMemoryBlock(array+0, array+1, sizeof(array)-1); // Upward, not allowed</code>
<b>Example 4</b>	<code>copyMemoryBlock( array+1, array+0, sizeof(array)-1 ); // Downward, allowed</code>



## moduleInfo

<b>Function</b>	Store Module data to <code>bufferINFO</code>																											
<b>Purpose</b>	Get information about transceiver module and OS																											
<b>Syntax</b>	<code>void moduleInfo ()</code>																											
<b>Parameters</b>	-																											
<b>Return value</b>	-																											
<b>Output values</b>	<p><code>bufferINFO[0 to 7]</code>:</p> <table border="1" data-bbox="279 515 1332 660"> <tr> <th>address in <code>bufferInfo</code></th> <th>7</th> <th>6</th> <th>5</th> <th>4</th> <th>3</th> <th>2</th> <th>1</th> <th>0</th> </tr> <tr> <th>meaning</th> <td colspan="2">OS build</td> <td>MCU type</td> <td>OS version</td> <td colspan="4">Serial number</td> </tr> <tr> <td></td> <td colspan="2"></td> <td></td> <td></td> <td colspan="4">Module ID</td> </tr> </table> <p>Serial number (Module ID): 4 B identification code unique for each TR module.</p> <p>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).</p> <p>MCU type:  3: PIC16F886  4: PIC16LF1938</p> <p>OS build: OS subversion for the manufacturer only.</p> <p><i>Example (all in hexadecimal):</i></p> <pre> [0] [1] [2] [3] [4] [5] [6] [7] bufferINFO[0-7] = 1C 10 00 01 31 04 39 11 </pre> <p>Meaning: Module ID = 0100101C, IQRF OS version 3.01D, PIC16LF1936, build 0x1139.</p>	address in <code>bufferInfo</code>	7	6	5	4	3	2	1	0	meaning	OS build		MCU type	OS version	Serial number									Module ID			
address in <code>bufferInfo</code>	7	6	5	4	3	2	1	0																				
meaning	OS build		MCU type	OS version	Serial number																							
					Module ID																							
<b>Preconditions</b>	-																											
<b>Remarks</b>	Module data can also be read by SPI master. See the IQRF SPI specification [5].																											
<b>Side effects</b>	<code>bufferINFO[8 to 63]</code> is modified.																											
<b>See also</b>	<code>appInfo</code>																											
<b>Example</b>	<pre> uns24 SN @ bufferInfo; uns8 OSv @ bufferInfo[4]; moduleInfo(); // Now SN == module serial number // and OSv == OS version </pre>																											

**appInfo**

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

**SPI**
**enableSPI**

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

**disableSPI**

<b>Function</b>	Switch SPI HW module off and configure SPI pins as I/Os
<b>Purpose</b>	Disable SPI communication
<b>Syntax</b>	<code>void disableSPI ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	SPI Status is switched to <i>SPI not active</i> .
<b>Preconditions</b>	–
<b>Remarks</b>	The PIC internal SPI hardware module is disabled and related pins (C5 to C8 or Q6, Q7, Q8 and Q11) are reconfigured as general I/Os. See SPI Implementation in IQRF TR modules [5] and example E07-SPI [10].
<b>Side effects</b>	<ul style="list-style-type: none"> <li>• The appropriate PIC pins are not restored to the state before <code>enableSPI</code> calling.</li> <li>• Current packet is lost by both sides if SPI communication is running on background at this moment.</li> </ul>
<b>See also</b>	<code>enableSPI</code> , <code>startSPI</code> , <code>stopSPI</code> , <code>getStatusSPI</code> , <code>restartSPI</code>
<b>Example</b>	See <code>getStatusSPI</code>

**startSPI**

<b>Function</b>	Indicate ready to Master.
<b>Purpose</b>	<ul style="list-style-type: none"> <li>Initiate SPI packet transmission from Slave (request to Master). Provide data from <code>bufferCOM</code> to Master according to Master's clock (on OS background).</li> <li><code>startSPI(0)</code> indicates to Master that the Slave is ready to receive data (<code>bufferCOM</code> not full).</li> </ul>
<b>Syntax</b>	<code>void startSPI (length)</code>
<b>Parameters</b>	<code>uns8 length</code> : number of bytes to be sent (0 to 64)
<b>Return value</b>	–
<b>Output values</b>	SPI Status is switched to: <ul style="list-style-type: none"> <li><i>SPI data ready</i> – after <code>startSPI(1 to 64)</code></li> <li><i>SPI ready, Communication mode</i> – after <code>startSPI(0)</code>.</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>SPI must be enabled by the <code>enableSPI</code> function before.</li> <li><code>startSPI</code> must not be combined with functions changing <code>bufferCOM</code>, e.g. <code>discovery</code></li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>SPI runs on OS background.</li> <li><code>startSPI(0)</code> is also useful for recovering SPI from communication failures (e.g. the CRC mismatch).</li> <li>See SPI Implementation in IQRF TR modules [5] and example E07-SPI [10].</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>enableSPI</code> , <code>disableSPI</code> , <code>stopSPI</code> , <code>getStatusSPI</code> , <code>restartSPI</code>
<b>Example 1</b>	<pre> // Slave -&gt; Master bufferCOM[0] = "I"; bufferCOM[1] = "Q"; enableSPI(); startSPI(2);           // Request to Master is active on background from now ...                   // and the program just continues here </pre>
<b>Example 2</b>	<code>startSPI(0);</code> // Reset SPI communication
<b>Example 3</b>	See <code>getStatusSPI</code>

### stopSPI

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

### restartSPI

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

### getStatusSPI

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

**RF**
**setTXpower**

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

**setRFspeed**

<b>Function</b>	Select RF bit rate
<b>Purpose</b>	Select RF bit rate
<b>Syntax</b>	<code>void setRFspeed (speed)</code>
<b>Parameters</b>	uns8 speed: <ul style="list-style-type: none"> <li>• 1 1.2 kb/s (preliminary, not for 433 MHz TR modules)</li> <li>• 2 19.2 kb/s (default)</li> <li>• 3 57.6 kb/s (preliminary, not for 433 MHz TR modules)</li> <li>• 4 86.2 kb/s (preliminary, not for 433 MHz TR modules)</li> </ul>
<b>Return value</b>	–
<b>Output values</b>	Available read only in the RFspeed register
<b>Preconditions</b>	Bit rates different from 19.2 kb/s are preliminary, for experimental purpose only.
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Non-default bit rates are provisionally intended for experimental purposes only.</li> <li>• Routing is supported for 19.2 kb/s only</li> </ul>
<b>Side effects</b>	RF channel must be specified after every bit rate change.
<b>See also</b>	setRFchannel
<b>Example 1</b>	<code>setRFspeed(1); // 1.2 kb/s selected</code> <code>setRFchannel(...); // channel must be selected then</code>
<b>Example 2</b>	<code>setRFspeed(2); // 19.2 kb/s selected</code> <code>setRFchannel(...); // channel must be selected then</code>

### setRFband

<b>Function</b>	Select RF frequency band
<b>Purpose</b>	Select 868 MHz or 916 MHz band
<b>Syntax</b>	<code>void setRFband (band)</code>
<b>Parameters</b>	uns8 band: <ul style="list-style-type: none"> <li>• 0 868 band MHz (default)</li> <li>• 1 916 band MHz</li> </ul>
<b>Return value</b>	–
<b>Output values</b>	Flag <code>_916MHz</code> in the <code>userInterface</code> register: <code>_916MHz</code> : 0 – 868 MHz band 1 – 916 MHz band
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• This function is not implemented for TR modules with fixed band, e.g. TR-54D-433.</li> <li>• To ensure compatibility with future OS versions, it is not recommended to use this function from OS v3.04D. RF band should be configured in IQRF IDE (<i>TR Configuration</i>).</li> </ul>
<b>Remarks</b>	Default RF band and channel after reset are set according to <i>TR Configuration</i> .
<b>Side effects</b>	RF channel must be specified after every band change.
<b>See also</b>	<code>setRFchannel</code>
<b>Example1</b>	<code>setRFband(1);                    // 916 MHz band selected</code>
<b>Example2</b>	<code>setRFband(0);                    // 868 MHz band selected</code>

### setRFchannel

<b>Function</b>	Set RF channel
<b>Purpose</b>	Select free RF channel for not interfered communication
<b>Syntax</b>	<code>void setRFchannel (channel)</code>
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• uns8 channel: see IQRF OS User's guide, Appendix 2, Channel map</li> <li>• Default:               <ul style="list-style-type: none"> <li>• 868 MHz band: 52</li> <li>• 916 MHz band: 104</li> <li>• 433 MHz band: 8</li> </ul> </li> </ul> Default channel can be changed by <i>TR Configuration</i> in IQRF IDE.
<b>Return value</b>	–
<b>Output values</b>	Available read only in the <code>RFchannel</code> register
<b>Preconditions</b>	–
<b>Remarks</b>	Channel 0 is reserved for DPA service purposes. It is not recommended to use it for regular communication.
<b>Side effects</b>	RF channel must be specified after every bit rate change.
<b>See also</b>	<code>setRFspeed</code>
<b>Example</b>	<code>setRFspeed(3);                    // 57.6 kb/s bit rate selected</code> <code>setRFchannel(25);                // 868.15 MHz channel selected</code>



**setRFmode**

<b>Function</b>	Set RF mode
<b>Purpose</b>	Specify RFRX and RFTX power modes, signal filtering modes for RFRX and enable fast response to external signal during LP/XLP RFRX.
<b>Syntax</b>	<code>void setRFmode (mode)</code>
<b>Parameters</b>	<p>uns8 mode: SWTTFFRR in binary</p> <ul style="list-style-type: none"> <li>• <b>S: Stay in RX mode</b> (for fast response for following <code>checkRF</code>, <code>RFRXpacket</code> or <code>RFTXpacket</code>) <ul style="list-style-type: none"> <li>1 RX chain stays enabled after <code>checkRF</code>, <code>RFRXpacket</code> and <code>RFTXpacket</code></li> <li>0 RX chain is disabled after <code>checkRF</code>, <code>RFRXpacket</code> and <code>RFTXpacket</code></li> </ul> </li> <li>• <b>W: Wait packet end</b> <ul style="list-style-type: none"> <li>1 Waits until receipt is finished if data payload (but not the preamble) is currently receiving, even though <code>toutRF</code> timeout is over meanwhile.</li> <li>0 <code>RFRXpacket</code> is unconditionally finished when <code>toutRF</code> timeout is over.</li> </ul> </li> <li>• <b>TT: TX mode</b> <ul style="list-style-type: none"> <li>00 for STD RX mode (standard preamble ~3 ms)</li> <li>01 for LP RX mode (prolonged preamble ~50 ms)</li> <li>10 for XLP RX mode (prolonged preamble ~900 ms)</li> <li>11 LP/XLP RX asynchronous termination on pin change enabled. If enabled and no packet is just received, low level on pin C5 (for TR modules in SIM format) or Q12 (for TR-54D) terminates RF reception in LP/XLP mode, immediately or after current cycle finishing. See examples 3 and 4.</li> </ul> </li> <li>• <b>FF: Filter incoming signal</b> in LP, XLP and RFIM RX modes (<code>RR</code> ≠ 0). Signal with lower level is ignored. Relative RF range is shortened due to this filtration. The level corresponds to the <code>checkRF(x)</code> parameter: <ul style="list-style-type: none"> <li>00 x = 5</li> <li>01 x = 20</li> <li>10 x = 35</li> <li>11 x = 50</li> </ul> </li> <li>• <b>RR: RX mode</b> <ul style="list-style-type: none"> <li>00 STD RX mode (Standard, transmitting device should have <code>TT=00</code>)</li> <li>01 LP RX mode (Low power, transmitting device should have <code>TT=01</code>)</li> <li>10 XLP RX mode (Extra low power, transmitting device should have <code>TT=10</code>)</li> <li>11 RFIM mode (<code>RFRXpacket</code> is terminated when signal strength falls below the <code>FF</code> level)</li> </ul> </li> </ul>
<b>Return value</b>	–
<b>Output values</b>	Available read only in the <code>RFmodeByte</code> register.
<b>Preconditions</b>	Non-STD RX modes are intended for bit rate 19.2 kb/s only.
<b>Remarks</b>	Default value is mode = 0. See example E10-RFMODE.
<b>Side effects</b>	RF circuitry and MCU is temporarily set to sleep during low power RX modes. Thus, all tasks running on OS background can be untimely canceled. To avoid this, use <code>setRFmode</code> after finishing background tasks. See Example 2.
<b>See also</b>	<code>checkRF</code>
<b>Example 1</b>	<pre> setRFmode(0b00000000); // RX: STD, no filtering                         // TX: for STD RX (standard preambles) setRFmode(0b00010001); // RX: LP, lowest filtering (5)                         // TX: for LP RX (prolonged preambles ~50 ms) setRFmode(0b00101110); // RX: XLP, highest filtering (50)                         // TX: for XLP RX (prolonged preambles ~900 ms) setRFmode(0b00000101); // RX: LP, low filtering (20)                         // TX: for STD RX (standard preambles) setRFmode(0b10001011); // RX: RFIM, high filtering (35), stay in RX mode                         // TX: for STD RX (standard preambles) setRFmode(0b01000000); // RX: STD, no filtering, wait packet end                         // TX: for STD RX (standard preambles) </pre>

<b>Example 2</b>	<pre> while (getStatusSPI()) // wait for finishing SPI on background     clrwdt(); disableSPI(); SWDTEN = 0;           // possibly disable watchdog for lower consumption setRFmode(0b00010001); // and go to LP mode then </pre>
<b>Example 3</b>	<pre> // RFRXpacket terminated after low level on C5/Q12 is detected and // current cycle is finished.  toutRF = 100;           // [in cycles], 1 cycle == ~770 ms  while (1) {     setRFmode(0x32);     // RX_XLP + LP/XLP RX termination     if (RFRXpacket())     {         ...     }      ...                  // Goes here after every 77 s (toutRF=100) or                         // if low level appears on the C5/Q12 pin                         // in a moment when RX XLP cycle is finished      if (buttonPressed)     {         ...         setRFmode(0);    // Set required TX preamble         RFTXpacket();     } } </pre>
<b>Example 4</b>	<pre> // RFRXpacket terminated immediately after low level on C5/Q12 is detected. // It is necessary to activate interrupt on change periodically.  toutRF = 100;           // [in cycles], 1 cycle == ~770 ms  while (1) {     setRFmode(0x32);     // RX_XLP + LP/XLP RX termination     writeToRAM(&amp;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     IOCFIE = 1;          // Interrupt on change enabled      writeToRAM(&amp;IOCBF, IOCBF &amp; 0xEF); // Clear interrupt on change flag.   // Instead of IOCBF.4=0;   // Bit IOCBF.4 cannot be accessed   // directly due to OS restriction.      if (RFRXpacket())     {         ...     }      ...                  // Goes here after every 77 s (toutRF=100) or                         // immediately if low level appears on the C5/Q12 pin      if (buttonPressed)     {         ...         setRFmode(0);    // Set required TX preamble         RFTXpacket();     } } </pre>

### checkRF

<b>Function</b>	Check incoming RF signal strength for specified level.
<b>Purpose</b>	Incoming RF signal detection to start RF receiving.
<b>Syntax</b>	<code>bit checkRF(level)</code>
<b>Parameters</b>	<p><code>uns8 level = DQI + RSSI_FILTER</code></p> <ul style="list-style-type: none"> <li>DQI (Data Quality Indicator): <ul style="list-style-type: none"> <li>0x80 DQI enabled</li> <li>0 DQI disabled</li> </ul> </li> </ul> <p>If DQI is enabled <code>checkRF</code> returns true when there is an extended probability that just the FSK modulation is detected. See the RF IC datasheet. This can help to distinguish noise from valid signal.</p> <ul style="list-style-type: none"> <li>RSSI_FILTER: 0 to 64</li> </ul> <p>Higher level requires stronger signal. Relative RF range is shortened due to this filtration according the datasheet of the TR module, Table 3. RSSI offset is 32, e.g. level 16 means a signal with RSSI &gt; 48.</p>
<b>Return value</b>	<ul style="list-style-type: none"> <li>0: Signal with specified level or higher not detected RSSI &lt; RSSI_FILTER, with respect to possible DQI</li> <li>1: Signal with specified level or higher detected RSSI &gt; RSSI_FILTER, with respect to possible DQI</li> </ul>
<b>Output values</b>	<ul style="list-style-type: none"> <li>Signal strength is also available as a relative value in the ADRESH (one of PIC SFR registers). Higher value means stronger signal.</li> <li>If 1 is returned the RF IC stays in RX mode (RX chain on)</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>This function is intended for STD and RFIM receive modes but not for LP and XLP.</li> <li>If DQI enabled <code>PORTA.6 = 1</code> must be set before every <code>checkRF</code> usage. See Example 4.</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>Higher level means lower sensitivity which requires stronger signal resulting in higher immunity against interferences but allows lower range – see TR datasheet, table Relative RF range vs. level.</li> <li>Checking takes: <ul style="list-style-type: none"> <li>~950 <math>\mu</math>s (without RF signal) or ~800 <math>\mu</math>s (with RF signal) if RX chain is off (bit S = 0 in <code>setRFmode</code>)</li> <li>~400 <math>\mu</math>s (with or without RF signal) if RX chain is on (bit S = 1 in <code>setRFmode</code>)</li> </ul> </li> <li>Checking consumes ~9.5 mA.</li> <li>This function is intended for fast response and power consumption reduction in STD RX mode.</li> <li>If RF IC is in sleep (after <code>reset</code>, <code>setRFsleep</code> or <code>iqrfSleep</code>), the first <code>checkRF</code> call wakes-up the RF IC but return and output values are not correct. For proper values <code>setRFready</code> should be called first. See examples 5 a 6.</li> </ul>
<b>Side effects</b>	A/D converter control registers are changed
<b>See also</b>	<code>setRFmode</code> , <code>RFRXpacket</code> , <code>getRSSI</code> , <code>setRFready</code> , <b>macro</b> <code>setRFRXchain</code> in <code>IQRF-macros.h</code>
<b>Example 1</b>	<pre> // Fast response receiving in STD mode if (checkRF(5)) // Detect signal with RSSI &gt; 37 {     if (RFRXpacket()) // Duration according to toutRF only if packet is sent.     {         // toutRF can be optimized for expected packet length.         ...     } } // Otherwise only ~1 ms is spent. ... time-critical section can be placed here </pre>
<b>Example 2</b>	<pre> if (checkRF(10)) // Detect signal with RSSI &gt; 42 ... </pre>
<b>Example 3</b>	<pre> // RF signal strength analyzer SWDTEN = 0; // disable watchdog while (1)     if (checkRF(3)) pulseLEDR(); // LED flash if signal level &gt;= 3 detected </pre>
<b>Example 4</b>	<pre> PORTA.6 = 1; // Necessary if using DQI if (checkRF(0x85)) // Detect signal with RSSI &gt; 37 using DQI ... </pre>

<b>Example 5</b>	<pre>iqrfSleep(); setRFready();           // RF IC wake-up for follow-up checkRF if (checkRF(5))     pulseLEDG(); else     pulseLEDR();</pre>
<b>Example 6</b>	<pre>setRFmode(_STAYRX); ... iqrfSleep(); setRFready();           // RF IC wake-up for follow-up checkRF setRFRXchain();        // Necessary. See IQRF-macros.h if (checkRF(5))     pulseLEDG(); else     pulseLEDR();</pre>

### getRSSI

<b>Function</b>	Get RSSI (received signal strength indication) value of incoming RF signal.
<b>Purpose</b>	Fast RF signal level detection
<b>Syntax</b>	uns8 <b>getRSSI</b> ()
<b>Parameters</b>	–
<b>Return value</b>	RSSI value. See the RF IC datasheet. Unlike <code>checkRF</code> , no offset is considered.
<b>Output values</b>	–
<b>Preconditions</b>	RF IC must be in RX Receiving mode (RX chain set on). See examples below.
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Takes 66 <math>\mu</math>s.</li> <li>• RSSI value is also available in <code>lastRSSI</code> register after successful receipt by <code>RFRXpacket</code>.</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>checkRF</code> , <code>RFRXpacket</code>
<b>Example 1</b>	<pre> // With setRFready setRFready(); // Not obligatory. However, the waitMS delay (see // below) must be adjusted with respect to this. setRFmode(_STAYRX); // RX chain will stay on after follow-up checkRF checkRF(0); // Another function to switch RX chain on // (e.g. RFRXpacket) can also be used. Parameter value // is meaningless here. waitMS(1); // Wait until RX chain is on. It depends on // setRFready usage, see Example 2. i = getRSSI(); </pre>
<b>Example 2</b>	<pre> // Without setRFready setRFmode(_STAYRX); // RX chain will stay on after follow-up checkRF checkRF(0); // Another function to switch RX chain on // (e.g. RFRXpacket) can also be used. Parameter value // is meaningless here. waitMS(3); // Wait until RX chain is on. It depends on // setRFready usage, see Example 1. i = getRSSI(); </pre>
<b>Example 3</b>	<pre> // Fast three-channel RF scanner uns8 ch1_rssi, ch2_rssi, ch3_rssi; setRFmode(_STAYRX); // RX chain will stay on after follow-up checkRF checkRF(0); // To switch RX chain on waitMS(3); // Wait until RX chain is on setRFchannel(1); ch1_rssi = getRSSI(); setRFchannel(2); ch2_rssi = getRSSI(); setRFchannel(3); ch3_rssi = getRSSI(); </pre>

## RFTXpacket

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

<b>Example 1</b>	<pre> // Peer-to-peer topology PIN=0; // Peer-to-peer (update also after every RFRXpacket // before every RFTXpacket)  setNonetMode(); bufferRF[0] = "I"; // Data to send bufferRF[1] = "Q"; DLEN = 2; // 2 B packet RFTXpacket(); // Send the packet to all Peer-to-peer Nodes in range // and to all IQMESH Nodes having set filtering off // Program stays here until the packet is sent ... // and then continues </pre>
<b>Example 2</b>	<pre> // IQMESH without routing, packet from Coordinator to Node #10 PIN = 0; // PIN preclearing (update also after every RFRXpacket // before every RFTXpacket)  setCoordinatorMode(); // The NTWF flag (PIN.7) is set here. bufferRF[0] = "I"; // Data to send bufferRF[1] = "Q"; DLEN = 2; // 2 B packet RX = 10; // Packet for Node #10 // _ROUTEF = 0; // Routing disabled - not necessary (default by OS) RFTXpacket(); // Send the packet to IQMESH Node #10 in this network // Reception depends on the Node (its current network // or filtering) </pre>
<b>Example 3</b>	<pre> // IQMESH with routing // Packet from Coordinator to Node #10 PIN = 0; // PIN preclearing (update also after every RFRXpacket // before every RFTXpacket)  setCoordinatorMode(); // The NTWF flag (PIN.7) is set here. bufferRF[0] = "I"; // Data to send bufferRF[1] = "Q"; DLEN = 5; // 5 B packet RX = 10; // Packet for Node #10 _ROUTEF = 1; // Routing enabled for outgoing packets RTDEF = 1; // SFM (Static Full MESH) // RTDEF = 2; // DFM (Discovered Full MESH) RTDT0 = 10; // 10 hops // RTDT0 = eeReadByte[0]; // # hops = # bonded nodes  RTDT1 = 2; // Time slot = 2 ticks (20 ms is enough for DLEN=5) RFTXpacket(); // Send the packet to IQMESH Node #10 in this network // Reception depends on the Node (its current network // or filtering) </pre>

### RFRXpacket

<b>Function</b>	Receive RF packet to <code>bufferRF</code> and provide related information
<b>Purpose</b>	RF receiving
<b>Syntax</b>	bit <code>RFRXpacket ()</code>
<b>Parameters</b>	–
<b>Return value</b>	<ul style="list-style-type: none"> <li>• 1 – packet received</li> <li>• 0 – packet not received</li> </ul>
<b>Output values</b>	<ul style="list-style-type: none"> <li>• <code>lastRSSI</code> – the RSSI value after successful receipt (single sample). Quiet level (noise) is 25 - 28.</li> <li>• <code>DLEN</code> = packet length. This variable is destroyed if the receipt is not successful.</li> <li>• <code>PIN</code> is updated according to packet received. This variable is destroyed if the receipt is not successful.</li> <li>• <code>_NTWPACKET</code>: valid if <code>RFRXpacket</code> return value == 1 only: <ul style="list-style-type: none"> <li>• 1 – networking packet received</li> <li>• 0 – non-networking packet received</li> </ul> </li> <li>• Other related networking information in case of IQMESH.</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• Timeout should be specified in <code>toutRF</code> (1 to 255) in number of 10 ms ticks or for LP and XLP modes in cycles, see IQRF OS User's guide, RF RX and TX modes).</li> <li>• Peer-to-peer topology: nothing else</li> <li>• IQMESH: network related parameters (filtering, ...) should be predefined</li> </ul> See IQRF OS User's guide [1] and IQMESH specification [4].
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Unlike SPI, RF communication does not run on OS background. This function is active on foreground until the packet is received or timeout expired. Timeout during packet receiving terminates the reception except of the Wait packet end mode – see <code>setRFmode</code>.</li> <li>• If the packet is sent when the addresse (or a routing device) is not executing this function the packet is lost.</li> <li>• Peer-to-peer topology: All non-networking packets in range are received.</li> <li>• IQMESH: Device receives only packets intended for it and non-networking packets depending on filtering mode – see <code>setNetworkFilteringOn</code> and <code>setNetworkFilteringOff</code>.</li> <li>• After termination in LP mode, RF IC is switched to RF ready mode.</li> <li>• After termination in XLP mode, RF IC is switched to RF sleep mode.</li> <li>• <code>RFRXpacket</code> is allowed to be called at least 5 ms after <code>RFTXpacket</code> in LP and XLP modes. See Example 4.</li> <li>• See examples E02–RX, E03–TR, E09–LINK, E11-IQMESH-N and E14-CONSUMPTION [10].</li> </ul>
<b>Side effects</b>	<ul style="list-style-type: none"> <li>• Update <code>PIN</code> before every <code>RFTXpacket</code> followed after <code>RFRXpacket</code>.</li> <li>• Result of <code>captureTicks</code> is destroyed if <code>startCapture</code> is active on background at the same time.</li> <li>• System tick timing is slightly affected.</li> <li>• <code>bufferRF[DLEN]</code> and <code>bufferRF[DLEN+1]</code> is destroyed.</li> <li>• The RF circuitry wakes up (in case of sleeping).</li> <li>• If a packet received the A/D converter control registers are changed.</li> </ul>
<b>See also</b>	<code>RFTXpacket</code> , <code>setRFmode</code> , <code>checkRF</code> and (in case of IQMESH) also other RF functions



<b>Example 1</b>	<pre> // Peer-to-peer topology toutRF = 10;           // RF timeout 100 ms if (RFRXpacket())    // Try to receive RF packet.                     // Program stays here until the packet is received                     // or the timeout is expired. Packet received? {     copyBufferRF2INFO(); // Yes:     PacketLength = DLEN; // Store received data                         // and possibly other info (packet length, ...) } else {     // No:     ...                // Timeout expired. Arrange respective operations. } </pre>
<b>Example 2</b>	<b>IQMESH: See setNodeMode and setNetworkFilteringOn.</b>
<b>Example 3</b>	<pre> if (RFRXpacket()) {     if (_ROUTEF)      // Was the packet routed?     {         while (RTDT0) // Yes - wait for finish of routing         {             waitDelay(RTDT1); // RTDT1 - timeslot             RTDT0--;          // Do not answer until all hops are finished         }     }     ...                // Now the Node is allowed to answer } </pre>
<b>Example 4</b>	<pre> setRFmode(_RX_LP);   // or _RX_XLP ... RFTXpacket(); ... waitMS(5);           // If there is no other code taking at least 5 ms,                     // the delay must be included here if (RFRXpacket()) ... </pre>

## Networking

### setCoordinatorMode

<b>Function</b>	Set Coordinator mode
<b>Purpose</b>	Assign the TR module as a network Coordinator
<b>Syntax</b>	<code>void setCoordinatorMode ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	<ul style="list-style-type: none"> <li>• Flag <code>_networkingMode (userInterface.7) = 1</code></li> <li>• Flag <code>_networkTwo (userInterface.6) = 0</code></li> <li>• In Coordinator mode the <code>_NTWF</code> flag (PIN.7) is automatically set before calling <code>RFTXpacket</code></li> </ul>
<b>Preconditions</b>	For IQMESH only
<b>Remarks</b>	Every TR module can work as a Coordinator or a Node. Just one Coordinator in single network is allowed. Avoid dynamic switching the Coordinator from device to device in a network. This settings affects both <code>RFRXpacket</code> and <code>RFTXpacket</code> .
<b>Side effects</b>	–
<b>See also</b>	<code>setNodeMode</code> , <code>setNonetMode</code> , <code>RFTXpacket</code>
<b>Example</b>	

### setNodeMode

<b>Function</b>	Set Node mode
<b>Purpose</b>	Assign the TR module as a network Node
<b>Syntax</b>	<code>void setNodeMode ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	<ul style="list-style-type: none"> <li>• Flag <code>_networkingMode (userInterface.7) = 1</code></li> <li>• Flag <code>_networkTwo (userInterface.6) = 1</code></li> <li>• In Node mode the <code>_NTWF</code> flag (PIN.7) is automatically set before calling <code>RFTXpacket</code></li> </ul>
<b>Preconditions</b>	For IQMESH only
<b>Remarks</b>	Every TR module can work as a Coordinator or a Node. This settings affects both <code>RFRXpacket</code> and <code>RFTXpacket</code> .
<b>Side effects</b>	–
<b>See also</b>	<code>setCoordinatorMode</code> , <code>setNonetMode</code> , <code>RFTXpacket</code>
<b>Example</b>	

### setNonetMode

<b>Function</b>	Select Peer-to-peer mode
<b>Purpose</b>	Switch from IQMESH to Peer-to-peer
<b>Syntax</b>	void <b>setNonetMode</b> ()
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	• Flag <code>_networkingMode</code> ( <code>userInterface.7</code> ) = 0
<b>Preconditions</b>	–
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Default OS mode is Peer-to-peer.</li> <li>• This settings affects <code>RFRXpacket</code> and <code>RFTXpacket</code> features.</li> <li>• PIN is not affected immediately but it is cleared after subsequent <code>RFRXpacket</code> or <code>RFTXpacket</code>.</li> <li>• Flag <code>_networkTwo</code> (<code>userInterface.6</code>) is not changed.</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>setCoordinatorMode</code> , <code>setNodeMode</code>
<b>Example</b>	<pre> setNetworkOne();           // TR communicates in IQMESH networking mode here ...                       // setNonetMode();           // Switch to Peer-to-peer mode ...                       // Now TR communicates without networking support </pre>

### setNetworkFilteringOn

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

### setNetworkFilteringOff

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

### setUserAddress

<b>Function</b>	Assign a user address to a Node
<b>Purpose</b>	User addressing of Nodes
<b>Syntax</b>	<code>void setUserAddress (address)</code>
<b>Parameters</b>	<code>uns16 address</code> : user address 1 to 65 000
<b>Return value</b>	–
<b>Output values</b>	–
<b>Preconditions</b>	For IQMESH Node and DFM2B only.
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• 0xFFFF is intended for broadcast.</li> <li>• Groups can be created by assigning the same address to more Nodes.</li> <li>• See Routing algorithms in the IQRF OS user's guide for details.</li> <li>• It is often convenient to set this as a part of bonding procedure by the user (to keep user program the same for all Nodes etc.).</li> <li>• Node User address is stored in EEPROM and is accessible via <code>getNetworkParams</code>. See Example 4.</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>bondNewNode</code>
<b>Example 1</b>	<code>setUserAddress(2000); // The Node has got user address 2000</code>
<b>Example 2</b>	<pre> setUserAddress(UA); eeWriteByte(E EUA, UA) // User address stored to EEPROM ... reset(); // User address lost after reset setUserAddress(eeReadByte(E EUA)); // User address restored from EEPROM </pre>
<b>Example 3</b>	<pre> getNetworkParams(); // Get User address uns16 myAddress = ntwUSERADDRESS; // See IQRF-memory.h </pre>

### getNetworkParams

<b>Function</b>	Get network parameters
<b>Purpose</b>	Get some information about current system, RF and network parameters
<b>Syntax</b>	<code>getNetworkParams ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	<ul style="list-style-type: none"> <li>• param2: Address of the device in network. <ul style="list-style-type: none"> <li>• 0                Unbonded device</li> <li>• 1 – 239        Bonded Node (logical address)</li> <li>• 254 (0xFE)    Prebonded Node</li> </ul> </li> <li>• bit <code>_NTWPACKET</code> <ul style="list-style-type: none"> <li>1 – IQMESH packet</li> <li>0 – Peer-to-peer packet</li> </ul> </li> <li>• param3: Network identification (<code>param3.high=NID1, param3.low=NID0</code>). If the device is bonded <code>NID0</code> and <code>NID1</code> refer to Coordinator otherwise to the device itself. These features are not guaranteed for future OS versions.</li> <li>• Network parameters (registers with names beginning with the <code>ntw</code> prefix) are updated. See IQRF OS User's guide, Appendix 2, table OS, RF and network parameters.</li> </ul>
<b>Preconditions</b>	For IQMESH only.
<b>Remarks</b>	See example E11 - IQMESH-N [10].
<b>Side effects</b>	–
<b>See also</b>	<code>amIBonded</code>
<b>Example</b>	<pre> if (amIBonded())           // Is the Node bonded? {     // Yes:     getNetworkParams();    // Get Node number     myAddr = param2; } </pre>

## Routing

### setRoutingOn

<b>Function</b>	Routing enabled
<b>Purpose</b>	Allow the Node to route packets on background.
<b>Syntax</b>	<code>void setRoutingOn ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	<ul style="list-style-type: none"> <li>• Enables to assign a VRN (Virtual Routing Number) to the Node during Discovery</li> <li>• <code>Flag_disableRouting = 0</code></li> <li>• This state is stored in EEPROM and initialized after reset.</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Nodes only</li> <li>• For DFM routing algorithms only</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Routing must be enabled for a Node to be assigned to the routing backbone during Discovery.</li> <li>• Routing can be enabled in all receive modes (STD, LP, XLP and RFIM).</li> <li>• <code>Flag_disableRouting</code> in register <code>_ntwCFG</code> is available read only after calling <code>getNetworkParams</code>:  <code>_disabledRouting: 0 - Routing on</code>  <code>                  1 - Routing off</code></li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>setRoutingOff</code> , <code>discovery</code> , <code>isDiscoveredNode</code> , <code>wasRouted</code>
<b>Example</b>	–

### setRoutingOff

<b>Function</b>	Routing disabled
<b>Purpose</b>	Forbid the Node to route packets on background.
<b>Syntax</b>	<code>void setRoutingOff ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	<ul style="list-style-type: none"> <li>• Disables to assign a VRN (Virtual Routing Number) to the Node during Discovery</li> <li>• <code>Flag_disableRouting = 1</code></li> <li>• This state is stored in EEPROM and initialized after reset.</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Nodes only</li> <li>• For DFM routing algorithms only</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If routing is disabled the Node will not be assigned to the routing backbone during Discovery.</li> <li>• <code>Flag_disableRouting</code> in register <code>_ntwCFG</code> is available read only after calling <code>getNetworkParams</code>:  <code>_disabledRouting: 0 - Routing on</code>  <code>                  1 - Routing off</code></li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>setRoutingOn</code> , <code>discovery</code> , <code>isDiscoveredNode</code> , <code>wasRouted</code>
<b>Example</b>	–

## discovery

<b>Function</b>	Discover Nodes for routing and assign VRN (Virtual Routing Number) to individual Nodes
<b>Purpose</b>	Routing backbone creation (for routing transparent from the user's point of view)
<b>Syntax</b>	<code>uns8 discovery (zones)</code>
<b>Parameters</b>	<code>uns8: zones: max. number of zones to be established</code>
<b>Return value</b>	Number of discovered Nodes ( $\leq$ number of Nodes which should route)
<b>Output values</b>	<ul style="list-style-type: none"> <li>Routing backbone is stored in EEPROM</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>For IQMESH Coordinator only.</li> <li>Must be performed in STD RX mode only.</li> <li>Nodes must be in the <code>answerSystemPacket</code> loop routine during Discovery.</li> <li>LED indication of passing discovery can be disabled (to save power consumption) or enabled (for development, service or demonstration) by the <code>_systemLEDindication</code> bit variable from OS v3.04D. Default is disabled (<code>_systemLEDindication=0</code>).</li> <li>SPI must not run in background during <code>discovery</code></li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>Nodes in current network only are discovered.</li> <li>Discovery should be invoked after every change in network topology.</li> <li>Nodes use the TX output power currently set in Coordinator during the discovery process.</li> <li>It is recommended to run <code>discovery</code> under stronger conditions than ones that will be used in normal communication. It can be achieved by lower RF power.</li> <li>See example E11-IQMESH-C [10].</li> <li>See IQRF OS User's guide, routing algorithms.</li> </ul>
<b>Side effects</b>	<ul style="list-style-type: none"> <li>Watchdog is disabled during this operation and enabled after finishing.</li> <li>All OS buffers (<code>bufferINFO</code>, <code>bufferCOM</code>, <code>bufferRF</code> and <code>bufferAUX</code>) are modified</li> <li><code>toutRF</code> variable is changed</li> <li>STAY_RX flag and other RFmode related functionality may be changed. See <code>setRFmode</code>.</li> <li>All OS registers regarding RF communication (relating to network parameters sent in the packet, e.g. <code>RX</code> and <code>RTDT0 - RTDT3</code>) may be changed.</li> <li>A/D converter control registers are changed</li> </ul>
<b>See also</b>	<code>setRoutingOn</code> , <code>setRoutingOff</code> , <code>isDiscoveredNode</code> , <code>bondNewNode</code> , <code>answerSystemPacket</code>
<b>Example 1</b>	<pre> setTXpower(DISCOVERY_POWER);           // Set RF power for discovery setRFmode(_RX_STD);                     // Set STD RX mode nodes = discovery(10);                   // Limit to max. 10 zones SWDTEN = 0;                              // Possibly restore WDT setRFmode(MY_RFMODE);                    // Restore RF mode parameters setTXpower(MY_POWER);                    // Restore RF power </pre>
<b>Example 2</b>	<pre> nodes = discovery(eeReadByte(0x00));      // Limit to number of bonded Nodes </pre>



### answerSystemPacket

<b>Function</b>	Enable response to Coordinator for Discovery and Node authorization
<b>Purpose</b>	Discovery and Node authorization (during remote bonding) support from the Node's side
<b>Syntax</b>	<code>void answerSystemPacket ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	System information exchanged between Coordinator and the Node via system packets.
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Node only.</li> <li>• Must be performed in STD RX mode only.</li> <li>• Nodes must be in the <code>answerSystemPacket</code> loop routine when Discovery or Node authorization is running.</li> <li>• WDT should be disabled before <code>answerSystemPacket</code></li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Nodes use the TX output power currently set in Coordinator for discovery.</li> <li>• It is recommended to run <code>discovery</code> under stronger conditions than ones that will be used in normal communication. It can be achieved by lower RF power.</li> </ul>
<b>Side effects</b>	<ul style="list-style-type: none"> <li>• <code>toutRF</code> is modified</li> <li>• TX power can be affected during Discovery process</li> <li>• <code>STAY_RX</code> flag and other RFmode related functionality may be changed. See <code>setRFmode</code>.</li> <li>• All OS registers regarding RF communication (relating to network parameters sent in the packet, e.g. <code>RX</code> and <code>RTDT0 – RTDT3</code>) may be changed.</li> <li>• A/D converter control registers are modified.</li> </ul>
<b>See also</b>	<code>setRoutingOn</code> , <code>setRoutingOff</code> , <code>isDiscoveredNode</code> , <code>discovery</code> , <code>nodeAuthorization</code>
<b>Example</b>	<pre> toutRF = MY_TOUT_RF; if (RFRXpacket ()) {     ... } else {     setWDToff ();     answerSystemPacket (); // to be discovered     setWDTon ();     setTXpower (MY_POWER); // Restore } </pre>

### isDiscoveredNode

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

### wasRouted

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

### optimizeHops

<b>Function</b>	Optimize number of hops for given Node
<b>Purpose</b>	Set optimized number of hops according to a topology, without flooding
<b>Syntax</b>	<code>void optimizeHops (x)</code>
<b>Parameters</b>	<p>uns8 x: optimizing method</p> <ul style="list-style-type: none"> <li>• 0xFF      DOM – Discovered optimized MESH: sets RTDT0 to VRN of addressed Node</li> <li>• 0x00      DRM – Discovered reduced MESH: sets RTDT0 to VRN of the first Node in the zone of the addressed Node.</li> </ul>
<b>Return value</b>	–
<b>Output values</b>	RTDT0 (number of hops) is set
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Coordinator and DFM routing algorithm only.</li> <li>• Intended to be called before sending a packet from Coordinator.</li> <li>• Node address must be set before (RX = ...).</li> <li>• The Node must be discovered.</li> </ul>
<b>Remarks</b>	See E11-IQMESH-C [10] and IQRF OS User's guide.
<b>Side effects</b>	–
<b>See also</b>	discovery, isDiscoveredNode
<b>Example</b>	<pre> setCoordinatorMode (); RX = MY_NODE; RTDT0 = eeReadByte(0x00);      // Hops according to a number of bonded Nodes     if (isDiscoveredNode(RX))  // For routing using Discovery only         optimizeHops(0xFF);    // Modifies RTDT0 (number of hops)     :     : RFTXpacket (); </pre>

## Bonding – Node only

### bondRequestAdvanced

<b>Function</b>	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 <i>Bonding</i> for more information.
<b>Purpose</b>	Request by the Node to be included to the network on both Coordinator's and Node's sides.
<b>Syntax</b>	bit <code>bondRequestAdvanced()</code>
<b>Parameters</b>	–
<b>Return value</b>	<ul style="list-style-type: none"> <li>• 1 – Node has been bonded</li> <li>• 0 – Node has not been bonded</li> </ul>
<b>Output values</b>	<ul style="list-style-type: none"> <li>• The <code>amIBonded</code> function starts to return value == 1 whenever is called while the Node is bonded by <code>bondRequestAdvanced</code> not beeing unbonded by <code>removeBond</code>.</li> <li>• Every <code>bondRequestAdvanced</code> calling increments the value of the internal counter (it is sent with the request). This counter is used with the <code>bondingMask</code> register to handle the situation when more than one Node with enabled prebonding would response to the request. See functions <code>bondNewNode</code> and <code>prebondNode</code> and IQRF User's guide, chapter <i>Bonding</i>. The counter is cleared after reset.</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Node only (<code>setNodeMode</code> must be called first)</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Bonding is a mutual relationship between Coordinator and Node. Coordinator assigns a Node number (1 to 239 or 0xFE) to the Node which serves as Node address within the network. (Coordinator itself has the address 0.) Bonding accomplishes via exchanging system RF packets and results are stored in system part of internal EEPROMs. The user can access results and change them via other bonding functions. See example E11-IQMESH-DFM-N, E11-IQMESH-DFM-C [10].</li> <li>• Prebonding is an initial phase of remote bonding. The new (bond requesting) Node gets the network ID and the universal address 0xFE from another (already bonded) Node. So the new Node becomes accessible in given network and can be authorized by the Coordinator to get a requested address.</li> <li>• This function takes cca 60 ms. It sends just one request for bonding and then waits for some time for the confirmation. This is the main difference between the recent <code>bondRequest</code> function which sends the request repeatedly for cca 10 s.</li> <li>• RF power and RF channel is not affected. STD mode is used.</li> <li>• The assigned address can be found out by function <code>getNetworkParams</code>.</li> </ul>
<b>Side effects</b>	<ul style="list-style-type: none"> <li>• <code>DLEN</code>, <code>PIN</code>, <code>toutRF</code>, <code>bufferRF</code> and <code>bufferINFO</code> are modified.</li> <li>• IQMESH mode must be restored by <code>setNodeMode</code> after <code>bondRequestAdvanced</code>.</li> <li>• A/D converter control registers are modified.</li> </ul>
<b>See also</b>	<code>bondNewNode</code> , <code>amIBonded</code> , <code>removeBond</code> , <code>rebondNode</code> , <code>getNetworkParams</code> , <code>setNodeMode</code> , <code>prebondNode</code> , <code>nodeAuthorization</code>
<b>Example</b>	<pre>while (!amIBonded())           // Request for beeing bonded (if not bonded yet) {     clrwdt();                  // If WDT active     pulseLEDG();     if (bondRequestAdvanced()) // Repeatedly try to bond         pulseLEDR();     waitDelay(1); }                               // Until successful setNodeMode();                // Restore</pre>

**bondRequest - obsolete**

<b>Function</b>	Ask Coordinator via RF for bonding to its network. Bond the Node in cooperation with Coordinator and record it to EEPROM.
<b>Purpose</b>	Request by the Node to be included to the network on both Coordinator's and Node's sides.
<b>Syntax</b>	bit <code>bondRequest ()</code>
<b>Parameters</b>	–
<b>Return value</b>	<ul style="list-style-type: none"> <li>• 1 – Node has been bonded</li> <li>• 0 – Node has not been bonded</li> </ul>
<b>Output values</b>	<ul style="list-style-type: none"> <li>• The <code>amIBonded</code> function starts to return value == 1 whenever is called while the Node is bonded by <code>bondRequest</code> not being unbonded by <code>removeBond</code>.</li> <li>• Coordinator is not affected at all.</li> <li>• <code>param2</code>: Node address (if successfully bonded only). Not guaranteed for future OS versions.</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Node only (<code>setNodeMode</code> must be called first)</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Bonding is a mutual relationship between Coordinator and Node. Coordinator assigns a Node number (1 to 239 or 0xFE) to the Node which serves as Node address within the network. (Coordinator itself has the address 0.) Bonding accomplishes via exchanging system RF packets and results are stored in system part of internal EEPROMs. The user can access results and change them via other functions related to bonding. See example E11 - IQMESH-N, E11 - IQMESH-C [10]. This function is active until successfully finished or fixed 10 s timeout expired. RF power is not affected.</li> </ul>
<b>Side effects</b>	<ul style="list-style-type: none"> <li>• <code>DLEN</code>, <code>PIN</code>, <code>toutRF</code>, <code>bufferRF</code> and <code>bufferINFO</code> are modified</li> <li>• Result of <code>captureTicks</code> is destroyed if <code>startCapture</code> is active on background at the same time.</li> <li>• Watchdog is disabled during this operation and enabled after finishing</li> <li>• IQMESH mode must be restored by <code>setNodeMode</code> after <code>bondRequest</code></li> <li>• A/D converter control registers are modified</li> </ul>
<b>See also</b>	<code>bondNewNode</code> , <code>amIBonded</code> , <code>removeBond</code> , <code>rebondNode</code> , <code>getNetworkParams</code> , <code>setNodeMode</code>
<b>Example 1</b>	<pre> pulsingLED();           // LED blinking indicates attempt to bond (max. 10 s) if (bondRequest()) {     stopLED();           // if successfully bonded     _RLED=1;             // LED On     waitDelay(100);      // for 1 s } setNodeMode();         // Restore stopLED(); </pre>
<b>Example 2</b>	See <code>amIBonded</code>

### prebondNode

<b>Function</b>	Look for bond requesting devices and prebond a new Node based on its request via RF. Assign the universal address 0xFE and Network ID and send both to the new Node via RF and get back Module ID (MID) of this new Node.
<b>Purpose</b>	Prebond a new Node to the network. This is the first phase of the remote bonding process and operates without direct action of the Coordinator. Then it is needed to deliver the obtained MID to the Coordinator (the second phase) by the application program. To complete remote bonding, the Coordinator must authorize the new Node (the third phase of the remote bonding process).
<b>Syntax</b>	bit <code>prebondNode ()</code>
<b>Parameters</b>	-
<b>Return value</b>	<ul style="list-style-type: none"> <li>• 1 – prebonding successful. <code>bufferINFO</code> contains MID of prebonded Node</li> <li>• 0 – prebonding unsuccessful. <code>bufferINFO</code> unchanged</li> </ul>
<b>Output values</b>	<ul style="list-style-type: none"> <li>• <code>bufferINFO[0 to 3]</code>: MID of the prebondedNode (if successfully prebonded), LSB in <code>bufferINFO[0]</code>.</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Node only (<code>setNodeMode</code> must be called first).</li> <li>• 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).</li> <li>• While the new Node is requesting to be prebonded (by function <code>bondRequestAdvanced</code>), the <code>prebondNode</code> function must be called in RF RX loop. See the Example below.</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• See example E11-IQMESH-DFM-N [10] and IQRF OS User's guide, chapter Bonding.</li> <li>• It is recommended to activate this function in application SW just in periods when it is actually needed. See the <code>prebondingEnabled</code> flag in the Example below. Permanent activation can ask for troubles with unwanted prebonding. If more Nodes have activated prebonding, it should be found out which Node has prebonded the new Node and ensure delivering of received MID to the Coordinator.</li> <li>• RF power and RF channel is not affected. The STD mode is used.</li> <li>• The function responds to bond request only if the following internal condition is valid:  <pre>if (((counter ^ address) &amp; bondingMask) == 0)</pre> <ul style="list-style-type: none"> <li>• <code>counter</code> Request for bonding contains a counter cleared after reset and incremented after every <code>bondRequestAdvanced()</code> call.</li> <li>• <code>address</code> Address of the supportive Node (evaluating the condition above).</li> <li>• <code>bondingMask</code> User-accessible register (with default value 0, i.e. the condition is true independently on the <code>counter</code> as well as the <code>address</code>). See IQRF User's guide, chapter Bonding for more information about the <code>bondingMask</code> usage.</li> </ul> </li> </ul>
<b>Side effects</b>	<p>The following values are modified and not restored:</p> <ul style="list-style-type: none"> <li>• PIN, DLEN, <code>bufferRF</code> and <code>bufferINFO</code> are modified</li> <li>• IQMESH mode must be restored by <code>setNodeMode</code> after <code>prebondNode</code></li> <li>• A/D converter control registers are modified</li> </ul>
<b>See also</b>	<code>bondRequestAdvanced</code> , <code>setNodeMode</code> , <code>nodeAuthorization</code> , <code>answerSystemPacket</code>
<b>Example</b>	<pre>if (RFRXpacket ()) {     ... } else {     // Try prebonding if enabled by user     if (prebondingEnabled &amp;&amp; prebondNode ())     {         setNodeMode ();           // Restore         ...                       // MID of prebonded Node is stored in bufferINFO     }     ... }</pre>

### amIBonded

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

### removeBond

<b>Function</b>	Remove the Node from the network and record it to EEPROM.
<b>Purpose</b>	Exclude the Node from the network on Node's side.
<b>Syntax</b>	void <b>removeBond</b> ()
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	<ul style="list-style-type: none"> <li>• The <code>amIBonded</code> function starts to return value == 0 whenever is called until the Node is bonded again via <code>bondRequestAdvanced</code>.</li> <li>• Coordinator is not affected at all.</li> </ul>
<b>Preconditions</b>	For IQMESH Node only ( <code>setNodeMode</code> must be called first).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• See example E11-IQMESH-DFM-N [10].</li> <li>• For rebonding use <code>bondRequestAdvanced</code> again.</li> <li>• <code>removeBond</code> relates to the Node only and <code>removeBondedNode</code> and <code>rebondNode</code> relate to Coordinator only. The other side is not informed by OS about changes made by these functions. If synchronization is needed it should be done by the application.</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>bondRequestAdvanced</code> , <code>bondNewNode</code> , <code>amIBonded</code> , <code>rebondNode</code>
<b>Example</b>	<code>removeBond(); // Remove the bond</code>

## Bonding – Coordinator only

### bondNewNode

<b>Function</b>	Local bonding. Looking for bond requesting devices and bond a new Node by Coordinator on a Node's request via RF in direct range. Allocate the Node number and assign the Network ID and send both to the Node via RF. If successful, the Node is bonded to the network on both Coordinator's and Node's sides and is included to the list of bonded Nodes provided by the Coordinator in EEPROM.
<b>Purpose</b>	Include a new Node in direct range to the network
<b>Syntax</b>	bit <code>bondNewNode (address)</code>
<b>Parameters</b>	<p>uns8: address</p> <ul style="list-style-type: none"> <li>• 1 to 239 Assign requested address to the Node. This must be unique in the whole network. If an existing number is used the Node is not bonded and the function immediately returns 0. Only these Nodes can be parts of routing backbone.</li> <li>• 0 The first free address is assigned. It equals to the number of bonded nodes + 1. It assumes a continuous block of addresses and possible vacations are ignored. Thus, this way is suitable for the initial bonding without discontinuities only.</li> <li>• 254 (0xFE) The universal address. Nodes with this address are included in the network but outside the routing backbone (not being discovered). Requested address can be assigned by <code>setUserAddress</code>. It is intended especially for networks with more than 239 Nodes.</li> </ul>
<b>Return value</b>	<ul style="list-style-type: none"> <li>• 1 – bonding successful, the Node is included to the list of bonded Nodes</li> <li>• 0 – bonding unsuccessful, the Node is not included to the list of bonded Nodes</li> </ul>
<b>Output values</b>	<ul style="list-style-type: none"> <li>• <code>param2</code>: Node number</li> <li>• <code>bufferRF[0 to 1]</code>: two lower ID bytes of the bonded Node, LSB in <code>bufferRF[0]</code>.</li> <li>• The <code>isBondedNode</code> function starts to return value == 1 whenever is called while the Node is in the list of bonded Nodes.</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Coordinator only. (<code>SetCoordinatorMode</code> must be called first.)</li> <li>• The Coordinator accomplishes bonding on request from a Node via RF. When this function is executing the <code>bondRequestAdvanced</code> function must be called in the Node.</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• See example E11-IQMESH-DFM-C [10] and IQRF OS User's guide, chapter Bonding.</li> <li>• If no requesting Node is detected during 10 s period this function terminates and returns 0.</li> <li>• Network ID is derived from Coordinator ID which ensures unique identification of various networks.</li> <li>• RF power and RF channel is not affected. The STD mode is used.</li> <li>• An occupied address can be unblocked by <code>removeBondedNode (address)</code>.</li> <li>• The function responds to bond request only if the following internal condition is valid:  <pre>if (((counter ^ address) &amp; bondingMask) == 0)</pre>           See <code>prebondNode</code> Remarks for description.</li> </ul>
<b>Side effects</b>	<p>The following values are modified and not restored:</p> <ul style="list-style-type: none"> <li>• <code>PIN</code>, <code>DLEN</code>, <code>toutRF</code>, <code>bufferRF</code> and <code>bufferINFO</code> are modified</li> <li>• Result of <code>captureTicks</code> is destroyed if <code>startCapture</code> is active on background at the same time.</li> <li>• Watchdog is disabled during this operation and enabled after finishing.</li> <li>• IQMESH mode must be restored by <code>setCoordinatorMode</code> after <code>bondNewNode</code></li> <li>• A/D converter control registers are modified</li> </ul>
<b>See also</b>	<code>bondRequestAdvanced</code> , <code>removeBondedNode</code> , <code>rebondNode</code> , <code>isBondedNode</code> , <code>setUserAddress</code> , <code>setCoordinatorMode</code> , <code>nodeAuthorization</code> , <code>prebondNode</code>
<b>Example</b>	<pre>if (bondNewNode(address)) // Bonding successful ? {     // Yes:     NodeNumber = param2;     ... } else {     // No:     ... // Arrange necessary steps } setCoordinatorMode(); // Restore</pre>



### nodeAuthorization

<b>Function</b>	Authorize the prebonded Node. Allocate the Node number and send it to the prebonded Node via RF network. If successful, the Node is bonded to the network on both Coordinator's and Node's sides and is included to the list of bonded Nodes provided by Coordinator in EEPROM.
<b>Purpose</b>	Include a new (prebonded) Node to the network. A last phase of the remote bonding process.
<b>Syntax</b>	bit <code>nodeAuthorization (mid, address)</code>
<b>Parameters</b>	<p>uns16 mid: 2 lower bytes of MID of prebonded Node to be authorized (HB = mid[0], LB = mid[1]).</p> <p>uns8 address:</p> <ul style="list-style-type: none"> <li>• 1 to 239 Assign requested address to the prebonded Node. This must be unique in the whole network. If an existing number is used the Node is not authorized and the function immediately returns 0. Only these Nodes can be parts of routing backbone.</li> <li>• 0 The first free address is assigned. It equals to the number of bonded nodes + 1. It assumes a continuous block of addresses and possible vacations are ignored. Thus, this way is suitable for the initial bonding without discontinuities.</li> <li>• 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 <code>setUserAddress</code>. It is intended especially for networks with more than 239 Nodes.</li> </ul>
<b>Return value</b>	<ul style="list-style-type: none"> <li>• 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 <code>removeBondedNode</code> and the authorization should be repeated.</li> <li>• 0 – Authorization unsuccessful, the requested address is already used. The Node is not included into the list of bonded Nodes.</li> </ul>
<b>Output values</b>	<ul style="list-style-type: none"> <li>• The <code>isBondedNode</code> function starts to return value == 1 whenever is called while the Node is in the list of bonded Nodes.</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Coordinator only. (<code>SetCoordinatorMode</code> must be called first.)</li> <li>• Authorized Node must already be prebonded.</li> <li>• Authorized Node must be in the <code>answerSystemPacket</code> loop routine during Athorization.</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• See example E11-IQMESH-DFM-C [10] and IQRF OS User's guide – Bonding.</li> <li>• This function sends an authorization packet (broadcast), does not wait for delivery and provides no checking. Therefore it is necessary to wait before sending the packet testing the result of the authorization till the authorization packet itself reaches the destination. This period to wait can be evaluated: number of routers in the network x time slot 50 ms.</li> <li>• An occupied address can be unblocked by <code>removeBondedNode (address)</code>.</li> <li>• RF power and RF channel is not affected. The STD mode is used.</li> </ul>
<b>Side effects</b>	<ul style="list-style-type: none"> <li>• PIN, DLEN, <code>bufferRF</code> and <code>bufferINFO</code> are modified</li> </ul>
<b>See also</b>	<code>bondNewNode</code> , <code>prebondNode</code> , <code>removeBondedNode</code> , <code>isBondedNode</code> , <code>setUserAddress</code> , <code>setCoordinatorMode</code> , <code>bondRequestAdvanced</code> , <code>answerSystemPacket</code>

<b>Example</b>	<pre> uns16 mid; mid.high8 = MID_0; mid.low8 = MID_1;  if (nodeAuthorization(mid, address)) // Was the address free? {     // Yes:     // Wait until authorization packet is delivered     ...     // 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         </pre>
----------------	---

### isBondedNode

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

### removeBondedNode

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

### rebondNode

<b>Function</b>	Put a Node back to the list of bonded Nodes by Coordinator in EEPROM
<b>Purpose</b>	Include the Node to the network again on Coordinator's side
<b>Syntax</b>	<code>bit rebondNode (n)</code>
<b>Parameters</b>	uns8 n: Node number
<b>Return value</b>	reserved for future OS versions
<b>Output values</b>	The <code>isBondedNode</code> function starts to return value == 1 whenever is called while the Node is in the list of bonded Nodes. The Node is not affected at all.
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• For IQMESH Coordinator only. (<code>SetCoordinatorMode</code> must be called first.)</li> <li>• Avoid rebonding a Node not being bonded ever before.</li> </ul>
<b>Remarks</b>	<code>removeBondedNode</code> and <code>rebondNode</code> relate to Coordinator only and <code>removeBond</code> relates to Node only. The other side is not informed by OS about changes made by these functions. If synchronization is needed it should be done by the application.
<b>Side effects</b>	–
<b>See also</b>	<code>bondNewNode</code> , <code>removeBondedNode</code> , <code>isBondedNode</code>
<b>Example</b>	<pre>rebondNode(28); // Coordinator assumes Node #28 to be                 // back in the network from now on</pre>

### clearAllBonds

<b>Function</b>	Remove all Nodes from the list of bonded Nodes by Coordinator in EEPROM
<b>Purpose</b>	Excluding all Nodes from the network on Coordinator's side
<b>Syntax</b>	<code>void clearAllBonds ()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	The <code>isBondedNode</code> function starts to return value <code>== 0</code> whenever is called while the Node is not in the list of bonded Nodes. Nodes are not affected at all.
<b>Preconditions</b>	For IQMESH Coordinator only. ( <code>SetCoordinatorMode</code> must be called first.)
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• See example E11-IQMESH-DFM-C [10].</li> <li>• After subsequent <code>bondNewNode(0)</code> the Coordinator will start to assign Node numbers from 0.</li> </ul>
<b>Side effects</b>	<code>bufferINFO</code> modified
<b>See also</b>	<code>removeBondedNode</code>
<b>Example</b>	<code>clearAllBonds(); // Exclude all currently bonded nodes from the network</code>

**RFPGM – wireless upload**
**enableRFPGM**

<b>Function</b>	Request to configure OS for switching to RFPGM mode after TR module reset
<b>Purpose</b>	Enable switching to RFPGM mode after reset
<b>Syntax</b>	<code>void enableRFPGM()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	IQRF OS is reconfigured. This function overrides the setting by <i>TR Configuration</i> performed in IQRF IDE.
<b>Preconditions</b>	
<b>Remarks</b>	This function must be executed first to modify OS and just the following reset will switch to RFPGM.
<b>Side effects</b>	–
<b>See also</b>	<code>disableRFPGM</code> , <code>runRFPGM</code> , <code>setupRFPGM</code>
<b>Example 1</b>	<pre>void APPLICATION() {   enableRFPGM();   ... }</pre>
<b>Example 2</b>	See <code>disableRFPGM</code>

**disableRFPGM**

<b>Function</b>	Request to configure OS for not switching to RFPGM mode after TR module reset
<b>Purpose</b>	Disable switching to RFPGM mode after reset
<b>Syntax</b>	<code>void disableRFPGM()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	IQRF OS is reconfigured. This function overrides the setting by <i>TR Configuration</i> performed in IQRF IDE.
<b>Preconditions</b>	–
<b>Remarks</b>	This function must be executed first to modify OS and just the following reset will not switch to RFPGM.
<b>Side effects</b>	–
<b>See also</b>	<code>enableRFPGM</code> , <code>setupRFPGM</code>
<b>Example 1</b>	<pre>enableRFPGM();           // During development // disableRFPGM();</pre>
<b>Example 2</b>	<pre>// enableRFPGM(); disableRFPGM();         // For final application</pre>

## runRFPGM

<b>Function</b>	Switch to RFPGM mode
<b>Purpose</b>	One-shot immediate switching to RFPGM mode
<b>Syntax</b>	<code>void runRFPGM()</code>
<b>Parameters</b>	–
<b>Return value</b>	–
<b>Output values</b>	RFPGM mode initiated
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• Non-networking modes and 19.2 kb/s bit rate must be selected.</li> <li>• Filtering of incoming packets must be disabled. (Use <code>setNetworkFilteringOff</code> if enabled.)</li> <li>• All user peripherals (UART, Timer6,...) used must have their interrupt enable flags (TXIE, TMR6IE,...) disabled first.</li> <li>• User interrupt must be disabled then. Observe the given order. See Example 2.</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• RF programming uses RF band and RF channel according to TR module configuration.</li> <li>• RFPGM mode can be refused: <ul style="list-style-type: none"> <li>• By low level on dedicated pin (if enabled). See <code>setupRFPGM</code> Parameters.</li> <li>• By the <i>End RFPGM</i> button in IQRF IDE (unconditionally)</li> <li>• ~1 minute after entering RFPGM mode (if enabled)</li> </ul> </li> <li>• After RFPGM finishing the application is always reset (regardless to RFPGM upload result).</li> <li>• After unsuccessful RFPGM upload the TR stays in RFPGM mode, see IQRF OS User's guide, Appendix 3 (RFPGM).</li> </ul>
<b>Side effects</b>	–
<b>See also</b>	<code>enableRFPGM</code> , <code>setupRFPGM</code>
<b>Example 1</b>	<pre>void APPLICATION() ... if (jumperSet) {     runRFPGM();           // Enter RFPGM mode on special request }</pre>
<b>Example 2</b>	<pre>RCIE = 0;                // All user peripheral interrupts must be TMR6IE = 0;              // disabled here (if used)                           // E.g. UART RX interrupt disable                           // E.g. Timer 6 interrupt disable  _enableUserInterrupt = 0; // Needed just if user interrupt is used                           // It must be called not until all user                           // peripheral interrupt enable flags are cleared,                           // see above.  setNonetMode();         // Disable networking (if enabled)  runRFPGM();             // Run on channel(s) according to TR configuration</pre>

## setupRFPGM

<b>Function</b>	Setup RFPGM parameters																		
<b>Purpose</b>	Configure RFPGM behavior																		
<b>Syntax</b>	<code>void setupRFPGM(x)</code>																		
<b>Parameters</b>	<p>uns8 x: <span style="float: right;">Factory default: 0x83</span></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">bit</th> <th style="width: 15%;">7</th> <th style="width: 15%;">6</th> <th style="width: 5%;">5</th> <th style="width: 5%;">4</th> <th style="width: 5%;">3</th> <th style="width: 5%;">2</th> <th style="width: 5%;">1</th> <th style="width: 5%;">0</th> </tr> </thead> <tbody> <tr> <td></td> <td>RFPGM termination by MCU pin(s)</td> <td>RFPGM termination after ~1 min</td> <td>0</td> <td>RFPGM enable</td> <td>0</td> <td>0</td> <td colspan="2">Single / dual channel</td> </tr> </tbody> </table> <p>Bit 0,1: RFPGM single / dual channel mode</p> <ul style="list-style-type: none"> <li>• 00 Receiving on single channel</li> <li>• 01 Reserved</li> <li>• 01 Reserved</li> <li>• 11 Receiving on dual channel (default, can be changed by <i>TR Configuration</i> in IQRF IDE)</li> </ul> <p>Bit 4: RFPGM invoking by reset. H – enabled, L – disabled (default). This bit operates like <code>enableRFPGM / disableRFPGM</code> functions.</p> <p>Bit 6: RFPGM termination automatically ~1 minute after entering RFPGM mode. H – enabled, L – disabled (default)</p> <p>Bit 7: RFPGM termination by MCU pins RA5 or RB4. H – enabled (default), L – disabled. If enabled, the termination is invoked by log. 0 for at least ~0.25 s for single channel or ~0.5 s for dual channel on one of the dedicated pin(s):</p> <ul style="list-style-type: none"> <li>• C5 for TR modules in SIM format, e.g. TR-52D</li> <li>• Q11 or Q12 for TR-54D</li> <li>• Q2 or Q3 for TR-55D</li> <li>• Q9 or Q12 for TR-56D</li> </ul> <p>Bits 2, 3, and 5: Must be kept cleared</p>	bit	7	6	5	4	3	2	1	0		RFPGM termination by MCU pin(s)	RFPGM termination after ~1 min	0	RFPGM enable	0	0	Single / dual channel	
bit	7	6	5	4	3	2	1	0											
	RFPGM termination by MCU pin(s)	RFPGM termination after ~1 min	0	RFPGM enable	0	0	Single / dual channel												
<b>Return value</b>	–																		
<b>Output values</b>	OS is modified and setup values are applicable anytime later.																		
<b>Preconditions</b>																			
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• RFPGM invoking by <code>runRFPGM()</code> is unconditional, independent on parameter x</li> <li>• RFPGM termination by IQRF IDE is unconditional, independent on parameter x</li> <li>• This function overrides the setting done by <i>TR Configuration</i> in IQRF IDE.</li> </ul>																		
<b>Side effects</b>	–																		
<b>See also</b>	<code>runRFPGM</code> , <code>enableRFPGM</code>																		
<b>Example 1</b>	<pre>void APPLICATION() setupRFPGM(0x13); // RFPGM entered: after reset or runRFPGM                   // RFPGM abandoned: by End RFPGM button only                   // Dual channel</pre>																		
<b>Example 2</b>	<pre>setupRFPGM(0x90); // RFPGM entered: after reset or runRFPGM                   // RFPGM abandoned: by dedicated pin or End RFPGM button                   // only                   // Single channel</pre>																		
<b>Example 3</b>	<pre>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</pre>																		

---

---

## Documentation and Information

- 1 IQRF OS User's guide [www.iqrf.org/155](http://www.iqrf.org/155)
- 2 **RAM map** and **EEPROM map**, IQRF OS User's guide, Appendix 1 [1]
- 3 **IQRF website** [www.iqrf.org](http://www.iqrf.org)
- 4 **IQMESH** specification [www.iqmesh.org/iqmesh](http://www.iqmesh.org/iqmesh)
- 5 IQRF **SPI** specification [www.iqrf.org/85](http://www.iqrf.org/85)
- 6 **IQRF support** [support@iqrf.org](mailto:support@iqrf.org)
- 7 **TR-52D** datasheet [www.iqrf.org/213](http://www.iqrf.org/213)  
**TR-54D** datasheet [www.iqrf.org/220](http://www.iqrf.org/220)  
**TR-55D** datasheet: [www.iqrf.org/239](http://www.iqrf.org/239)  
**TR-56D** datasheet: [www.iqrf.org/278](http://www.iqrf.org/278)
- 8 **PIC16LF1938** datasheet: [www.iqrf.org/214](http://www.iqrf.org/214)
- 9 **IQRF IDE**: [www.iqrf.org/86](http://www.iqrf.org/86)
- 10 **Examples** (included in the StartUp Package): [www.iqrf.org/112](http://www.iqrf.org/112)

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

## Document revision

- 131015 Remote bonding described. `startSPI(x)` and `discovery(x)` Preconditions updated.
- 130905 First release.



## Index

amiBonded.....	63	pulseLEDG.....	18
answerSystemPacket.....	57	pulseLEDR.....	16
appInfo.....	34	pulsingLEDG.....	17
bondNewNode.....	64	pulsingLEDR.....	16
bondRequest - obsolete.....	61	readFromRAM.....	22
bondRequestAdvanced.....	60	rebondNode.....	67
calibrateTimer.....	4	removeBond.....	63
captureTicks.....	12	removeBondedNode.....	67
checkRF.....	43	restartSPI.....	37
clearAllBonds.....	68	RFRXpacket.....	48
clearBufferINFO.....	31	RFTXpacket.....	46
clearBufferRF.....	31	runRFPGM.....	70
compareBufferINFO2RF.....	30	setCoordinatorMode.....	50
copyBufferCOM2INFO.....	29	setINDF0.....	24
copyBufferCOM2RF.....	29	setINDF1.....	24
copyBufferINFO2COM.....	27	setNetworkFilteringOff.....	52
copyBufferINFO2RF.....	27	setNetworkFilteringOn.....	52
copyBufferRF2COM.....	28	setNodeMode.....	50
copyBufferRF2INFO.....	28	setNonetMode.....	51
copyMemoryBlock.....	32	setOffPulsingLED.....	15
debug.....	7	setOnPulsingLED.....	15
disableRFPGM.....	69	setRFband.....	40
disableSPI.....	35	setRFchannel.....	40
discovery.....	56	setRFmode.....	41
eeeReadData.....	21	setRFready.....	6
eeeWriteData.....	21	setRFSleep.....	6
eeReadByte.....	19	setRFSpeed.....	39
eeReadData.....	19	setRoutingOff.....	55
eeWriteByte.....	20	setRoutingOn.....	55
eeWriteData.....	20	setTXpower.....	39
enableRFPGM.....	69	setupRFPGM.....	71
enableSPI.....	35	setUserAddress.....	53
getINDF0.....	25	startCapture.....	11
getINDF1.....	26	startDelay.....	13
getNetworkParams.....	54	startLongDelay.....	13
getRSSI.....	45	startSPI.....	36
getStatusSPI.....	38	stopLEDG.....	18
getSupplyVoltage.....	8	stopLEDR.....	17
getTemperature.....	9	stopSPI.....	37
iqrfSleep.....	5	swapBufferINFO.....	30
isBondedNode.....	66	waitDelay.....	10
isDelay.....	14	waitMS.....	10
isDiscoveredNode.....	58	waitNewTick.....	11
moduleInfo.....	33	wasRouted.....	58
optimizeHops.....	59	writeToRAM.....	23
prebondNode.....	62		

---

## Sales and Service

---

### Corporate office

MICRORISC s.r.o., Prumyslova 1275, 506 01 Jicin, Czech Republic, EU  
Tel: +420 493 538 125, Fax: +420 493 538 126, [www.microrisc.com](http://www.microrisc.com)

### Partners and distribution

please visit [www.iqrf.org/partners](http://www.iqrf.org/partners)

---

### Quality management

*ISO 9001 : 2009 certified*

### Trademarks

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

### Legal

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

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

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

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

---

<b>Website</b>	<b><a href="http://www.iqrf.org">www.iqrf.org</a></b>
<b>E-mail</b>	<b><a href="mailto:sales@iqrf.org">sales@iqrf.org</a></b>
<b>On-line support</b>	<b><a href="mailto:support@iqrf.org">support@iqrf.org</a></b>