IQRF OS

Operating System

Version 4.03D
for TR-7xD

User's Guide

Smarter Wireless. Simply.

Content

IQRF platform....3
Compatibility....3
IQRF OS versions and history....4
OS Principles....7
IQRF OS Architecture....8
OS plug-ins....9
OS upgrade....9
DPA....9
RF parameters....10
Microcontroller....11
MCU pins....12
Memories....13
Program memory (Flash)....15
Data memory (RAM)....16
Data memory (EEPROM inside the MCU)....19
Data memory (serial EEPROM)....19
Module identification....20
MID....20
IBK....20
TR configuration....21
OS....21
HWP....22
Security....22
Control....23
Operation modes....23
Real time....23
Watchdog....23
Power down modes....24
Sleep mode....24
Deep sleep mode....25
RF sleep mode....25
Other PIC peripherals....25
Reset....26
Interrupt....27
Temperature measurement....28
Battery check....28
LED indication....28
SPI....28
Debug....28
RF....29
RFcommunication....29
IQMESH packet....30
User interface....32
RF IC modes....33
RF transmitting....34
RF receiving....35
Low power modes....36
Filtering....37
Addressing....37
Routing....38
Fast Response Command....43
Bonding....45
Transceiver replacement....52
Backup....52
Restore....52
Encryption....53
Access encryption....53
Networking encryption....54
User encryption....54
IQMESH in practice....57
Routing explanation examples....57
Appendix 1 – Memory maps....61
RAM map (PIC16LF1983)....61
EEPROM map (inside the MCU, PIC16LF1983)....65
Flash map....66
Appendix 2 – Channel maps....67
433 MHz band channel map....67
868 MHz band channel map for TR-72D, TR-75D, TR-76D and TR-78D....68
868 MHz band channel map for TR-77D....69
916 MHz band channel map for TR-7xD....70
916 MHz band channel map for TR-7xD-IL....71
Appendix 3 – RFPGM - RF Programming™....72
Appendix 4 – Coordinator Bonding and Discovery Data....76
Bonded devices count....76
Bonded devices bitmap....76
Discovered devices bitmap....76
MID array....76
VRN array....76
Zones array....76
Parents array....77
Example....77
Internal EEPROM....77
Serial EEPROM....77
Appendix 5 – Migration from lower OS versions....78
Migration from OS v4.02D to v4.03D....78
Migration from OS v4.00D to v4.02D....78
Migration from OS v4.00D to v4.01D....78
Migration from OS v3.08D to v4.00D....79
Documentation and information....81
Document revision....81
Sales and Service....82

IQRF platform

IQRF is a wireless (RF) license free platform for ISM bands (868 MHz, 916 MHz and 433 MHz). Compact transceiver module (TR) has built-in operating system (OS) and is fully user programmable in C language using OS functions including RF (wireless) as well as SPI (4-wire serial) communication and complex IQMESH networking support. No link layer is provided, the entire functionality is fully up to user application. Optionally, the DPA (Direct Peripheral Access) framework utilizing HWP (Hardware Profiles) in the form of OS plug-ins instead of a user application can be used to implement a user-specific functionality without programming. Refer to DPA Framework Technical guide and DPA Quick Start guide.

Compatibility

<table>
<tr><th>TR module</th><th>Current OS</th><th>Modulation</th><th>IQRF IDE development environment</th></tr>
<tr><td>TR–11A</td><td rowspan="2">v2.08</td><td rowspan="4">ASK</td><td rowspan="4">v2.08</td></tr>
<tr><td>TR–21A v1.02 and v1.03</td></tr>
<tr><td>TR–31B</td><td rowspan="2">v2.10 for 3xB</td></tr>
<tr><td>TR–32B</td></tr>
<tr><td>TR–52B</td><td rowspan="2">v3.00</td><td rowspan="8">FSK</td><td rowspan="2">v2.08 possible, v4.xx [9] recommended</td></tr>
<tr><td>TR–53B</td></tr>
<tr><td>TR-52D</td><td rowspan="6">v3.08D</td><td rowspan="6">v4.xx
Use the latest version [9]</td></tr>
<tr><td>TR-53D</td></tr>
<tr><td>TR-54D</td></tr>
<tr><td>TR-55D</td></tr>
<tr><td>TR-56D</td></tr>
<tr><td>TR-58D-RJ</td></tr>
<tr><td>TR-72D</td><td rowspan="5">v4.02D</td><td rowspan="5">GFSK</td><td rowspan="5">v4.xx
Use the latest version [9]</td></tr>
<tr><td>TR-75D</td></tr>
<tr><td>TR-76D</td></tr>
<tr><td>TR-77D</td></tr>
<tr><td>TR-78D</td></tr>
</table>

Communication between transceivers with different RF modulation is absolutely not possible.

All TRs in the whole application must be equipped with the same IQRF OS version.

IQRF transceivers are normally delivered with current OS version. However, any lower OS version can be ordered on request. Therefore, it is not necessary to upgrade existing mature applications to current OS versions. Additionally, OS in IQRF transceivers can also be upgraded (or possibly downgraded) at the user. See chapter *OS upgrade*.

All IQRF transceivers support both OS and DPA approaches from IQRF OS v4.02D.

IQRF OS versions and history

The table below only illustrates the main differencies in OS versions. For exact changes refer to *Appendix 5*, *Migration from lower OS versions*.

Version	Main differences	Release	Status
v4.03D for 7xD	• **Smart connect** based on **IBK** (Individual Bonding Key) introduced • Dedicated fixed **service RF channels** • Operating **RF channel inherited** from network Coordinator • **4 B FRC** • User encryption slightly more flexible • LED control more user-friendly • Some minor improvements	Oct 2018	**Current** for **TR-72D**, **TR-75D**, **TR-76D**, **TR-77D** and **TR-78D**
v4.02D for 7xD	• TR and DCTR not differentiated. All TRs support both OS as well as DPA approaches. • TR upload extended. See *IQRF SPI Technical guide*. • Fixed a bug in 2B FRC individual packets processing for not discovered Nodes in range with the Coordinator only. • Higher immunity against possible failure of RF IC. • Some minor improvements	End of Aug 2017	Not for new designs for TR-72D, TR-75D, TR-76D, TR-77D and TR-78D
v4.01D for 7xD	• **Removed** due to a **serious bug** when using CATS based on TR with OS v4.01D to wirelessly upload TRs with OS v4.00D. A TR corrupted in this way can be **recovered** at the user. Contact IQRF support for instructions in this case.	14 Aug 2017	**Do not use**
v4.00D for 7xD	• **Packet structure** changed. • **Extended security** based on **AES-128** encryption • **Deep sleep** mode • Lots of **other enhancements** • DPA is the only recommended way for IQMESH implementation. • Networking with more than 240 devices is not supported. • DPA v3.00 support • IQRF **IDE v4.40** or higher required • `template-basic.h` header file renamed to `IQRF.h`	Mar 2017	Not for new designs for TR-72D and TR-76D

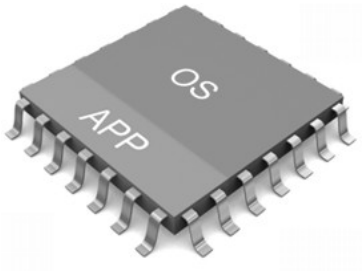
Version	Main differences	Release	Status
v3.09D for 7xD	• **TR-7xDx-IL** supported (intended **for Israel only**, with RF band/channel limitation).	May 2017	not for new designs for TR-72D and TR-76D
v3.08D for 7xD	• **433 and 916 MHz bands** implemented • **RF channel maps slightly changed**, more channels available • `isBondedNode() return` value slightly changed • `optimizeHops()` output value slightly changed • Minor bug and side effect fixed • DPA v2.26 to v2.28 support • IQRF I**DE v4.34** or higher required	Feb 2016	not for new designs for TR-72D and TR-76D
v3.08D for 5xD	• `isBondedNode() return` value slightly changed • `optimizeHops()` output value slightly changed • Minor bug fixed • DPA v2.26 to v2.28 support, final for TR-5xD. • IQRF **IDE v4.34** or higher required	Feb 2016	**final** for **TR-52D**, **TR-53D, TR-54D, TR-55D, TR-56D** and **TR-58-RJ**
v3.07D for 7xD	• **Different RF IC**, 916 MHz not yet supported • **MCU clock 16 MHz, faster SPI.** RF bit rate 19.8 kb/s. • **Selective FRC** and **2 B FRC** • `getSupplyVoltage`, `setRFmode`, `checkRF` and `getRSSI` changed • **Serial EEPROM: 32 KB**, more flexible access, error check • **OS upgradeable at the user** • Various slight improvements • DPA v2.20 to v2.24 support • IQRF **IDE v4.30** or higher required	Jul 2015	not for new designs for TR-72D and TR-76D
v3.07D for 5xD	• **Selective FRC and 2B FRC** • Serial EEPROM: more flexible access, error check • **OS upgradeable at the user** • Various slight improvements • DPA v2.20 to 2.24 support • IQRF **IDE v4.30** or higher required	Jul 2015	not for new designs for TR-52D, TR-53D, TR-54D, TR-55D, TR-56D and TR-58-RJ
v3.06D for 5xD	• Forced LP routing option added • **Advanced FRC** option added • Low power modes supported in bonding, FRC and Discovery • Various slight improvements and side effect removals • IQRF **IDE v4.20** or higher required	Nov 2014	not for new designs for TR-52D, TR-53D, TR-54D, TR-55D, TR-56D and TR-58-RJ
v3.05D for 5xD	• **Fast response command** (FRC) • Licensed Flash memory extended to 3008 machine instructions • Other enhancements • **DPA** v2.00 support • IQRF **IDE v4.15** or higher required	Apr 2014	not for new designs for TR-52D, TR-54D, TR-55D, TR-56D and TR-58-RJ
v3.04D for 5xD	• **TR configuration** (predefining OS parameters from IQRF IDE) • Enhanced **RFPGM** (**Standard** version instead of Lite version) • **Remote bonding** • A lot of additional improvements • **DPA** (Direct Peripheral Access) v1.00 support • IQRF **IDE v4.13** or higher required	Sep 2013	not for new designs for TR-52D, TR-54D, TR-55D, TR-56D and TR-58-RJ
v3.03D for 5xD	• Function **getRSSI**. • Some other improvements • IQRF **IDE v4.11** or higher required	Nov 2012	not for new designs for TR-52D, TR-54D, TR-55D and TR-56D
v3.02D for 5xD	• User interrupt, Timer6 user available • Lower power consumption in LP/XLP • Functions `setINDFx` and `getINDFx` • More precise timing • Improved Discovery • Series of other enhancements, see Appendix *Migration* • IQRF **IDE v4.05** or higher required	Sep 2012	not for new designs for TR-52D and TR-54D
v3.01D for 5xD	• **Powerful MCU** PIC16LF1938: **extended memories** (RAM 1024 B, Flash 16 kwords), **16-level stack**, efficient architecture, faster interrupt. • **Serial EEPROM** 16 Kb user available (not for Coordinator) • Several additional improvements	Apr 2012	not for new designs for TR-52D and TR-54D
v3.00 for 5xB	• Many new features, ~20 functions added • Up to 65 000 devices and up to 240 hops in IQMESH network • New power management modes, 35 uA in XLP RX • Discovery, real time transparent routing, low power routers • Various routing algorithms	Jan 2011	**final** for **TR-52B** and **TR-53B**

Version	Main differences	Release	Status
v2.11 for 5xB	• RF power management supported (`setRFmode`, `checkRF`, …) • RF channels available • Selectable RF bit rate (provisionally for experimental purpose)	Mar 2010	not for new designs for TR-52B and TR-53B
v2.10 for 5xB	In addition to that for 3xB: • 868 MHz or 916 MHz **band** software selectable • Enhanced battery check • RF IC sleep mode supported	Jan 2010	for TR-52B and TR-53B not for new designs
v2.10 for 3xB	• Concept of OS **plug-ins** • RF power not limited during bonding • Green LED support, LED functions renamed • User **RAM limited** to `0x1CF`	Dec 2009	**current** for **TR-31B** and **TR-32B**
v2.09	• Minor change in first falling to Sleep mode • Bonding robustness increased	Jul 2009	not for new designs
v2.08	• **broadcast** message support added	Oct 2008	**current** for **TR-11A** and **TR-21A**
	• Implemented in **TR-31B** modules	Nov 2008	
v2.07	• Bug in the **setLoggingOff()** function fixed • **Wake-up on pin change** improved. To utilize it, the sequence GIE = 0; RBIE = 1; is required just before iqrfSleep().	Sep 2008	not for new designs
v2.06	• Minor change in routing	Aug 2008	not for new designs
v2.05	• Higher **RF noise immunity** • Corrected transfer of MPRWx while not routing • Several minor bugs not affecting module functionality corrected	Aug 2008	not for new designs
v2.04	• **setNetworkFilteringOn**() switches just packet from active network (1 or 2), non-networking communication ignored • **Wake-up on pin change** under user's control. Default disabled. To enable, set RBIE = 1 before iqrfSleep(). Not compatible with previous versions (permanently enabled in Sleep up to v2.03).	Jul 2008	internal release only
v2.03	• BufferCOM size increased from 35B to 41B • Number of nodes in one network increased from 128 to 239 • Minor bug in routing fixed	Jul 2008	not for new designs
v2.02	• Minor SPI bug fixed	May 2008	not for new designs
v2.01	• Function **wipeBondNR**() added • Function **batteryValueOK**() added	Mar 2008	not for new designs
v2.00	• Much more **effective**, **easier** to use, higher **performance** • **Networking** totally reworked. Extended capability. **Complete IQMESH**. • **SPI on background** • **Encoded** network communication • Indirect RAM access • Temperature measurement supported by OS • Supports user application debugging directly by IQRF OS • Many other improvements • **IDE** – complete **development environment** with all SW tools integrated including effective debug tools	Jan 2008	not for new designs
v1.14	Previous generation	Jul 2007	not for new designs

OS Principles

The IQRF system is designed to allow using of RF wireless connection according to user's needs. Transceiver modules contain microcontrollers for controlling the transceiver operations and for executing of user defined functioning.

Patented IQRF transceiver module architecture has two software layers:



- Basic routines programmed in advance by the manufacturer. The set of such functions is called **operating system** (OS).
- **Application layer** utilizes routines from the basic layer to customize the module for user specific operation.

In opposite to Solution stack, there is no need to compile protocol related routines, just the application is compiled. This approach significantly reduces time and development costs.

OS offers software functions prepared in advance for all common user requirements. Thus, it is not necessary to create the whole user program by oneself (using microcontroller instructions and C commands only) but the user adds a user part of software to the OS only.

The user application so called „runs under the operating system" which means that this is invoked from OS, uses OS functions and is (should be) under OS control.

OS functions need not run sequentially (next function invoked not until the preceding one is finished) but some operations can run so called "in background" (the function arranges execution of requested operations which runs independently and immediately returns the control back to superior program). In this way more processes can run "simultaneously". Then the program structure is that besides of execution running sequentially "in foreground" several tasks in background can be running. IQRF OS allows to run even very complex operation including complete SPI communication protocol in background. This makes real-time programming really easy.

IQRF OS supports communications:

- **RF** (radio), including networks – in **peer-to-peer** and **IQMESH** topologies.
- Standard serial **SPI** (slave mode) interface for connection to peripherals or to PC (e.g. via CK-USB-04x).

Other communications can be realized with a user program (I^2C, UART, ...) utilizing HW communication modules inside the MCU.

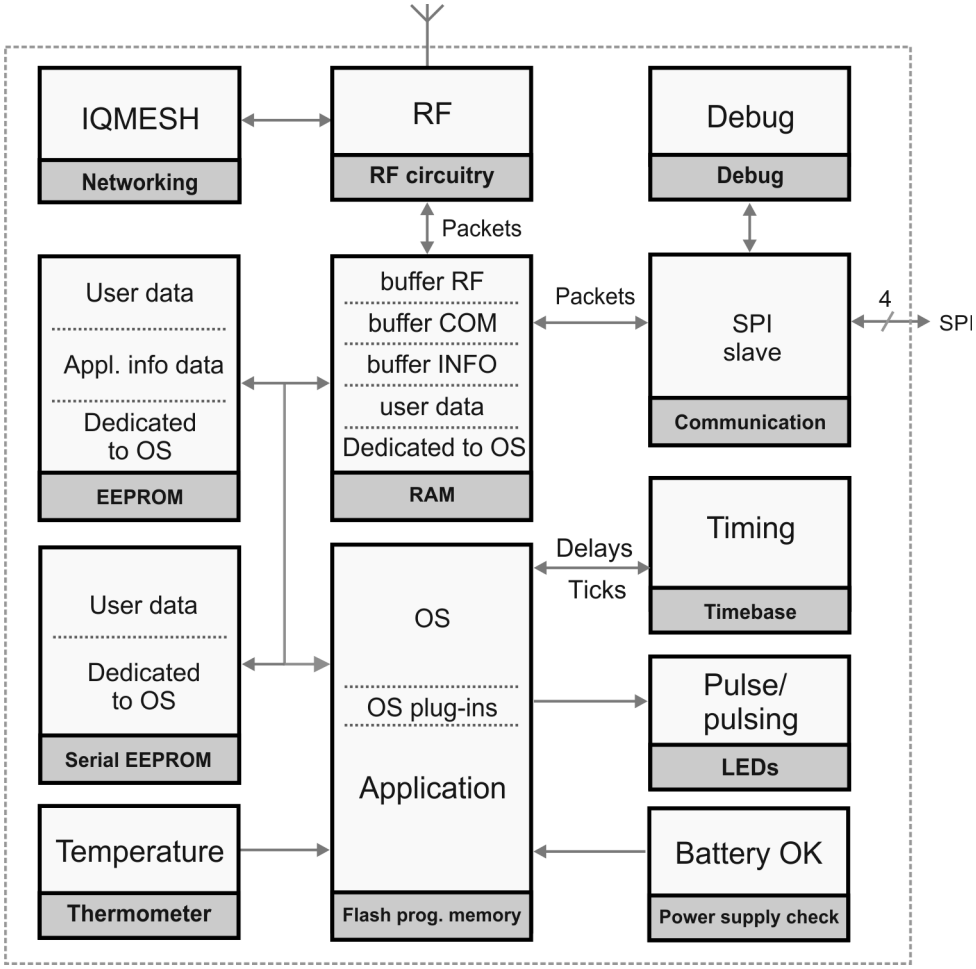
Complex standard communication interfaces (**USB**, **Ethernet**, **GSM**, …) can not be realized by a single TR module but using IQRF gateways.

OS supports low power consumption of IQRF transceivers with the **Sleep** functions when operation of TR module or RF IC is reduced/stopped. Power management RF modes optimize consumption when RF is active.

To increase the reliability the **watchdog** function is used. This is implemented in microcontroller hardware and controlled via user program.

IQRF OS Architecture

Hardware of the transceiver module with a microcontroller including internal resources and the IQRF OS creates the following architectural model:



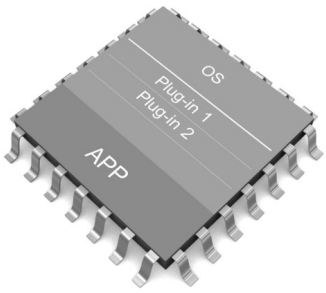
Individual blocks:

- Memories:
  - Program memory (Flash – inside MCU)
  - Data memory (RAM – inside MCU)
  - Data memory (EEPROM – inside MCU)
  - Data memory (EEPROM – external serial)
- Communication interface:
  - RF (wireless)
  - SPI (standard serial, 4-wire, slave, running in OS background)
- Temperature sensor (optional)
- Power supply check
- Digital I/O (input/output) – see datasheet of given TR transceiver.
  - SW selectable pull-up is available on pin RB4.
  - Wake-up from sleep and interrupt on pin change is available on pin RB4.
- A/D converter
- D/A converter – available on TR transceivers with not shared MCU pins on the Cx SIM pads.
- Analog comparator
- PWM output
- Timing support
  - 10 ms interval (tick) generator in background and supporting functions
  - Timer6 HW timer – see the MCU datasheet [8].
- 2 LEDs control in OS background
- IQMESH networking
- Debug: OS support for testing and debugging

Resources partially depend on transceiver module type.

OS plug-ins

IQRF operating system can be extended via optional plug-ins.



Plug-in is a SW module delivered (typically by the IQRF manufacturer, but can also be created by the user in collaboration with the IQRF manufacturer) as a file with the `.IQRF` extension. It should be uploaded to the TR module by the IQRF IDE and an IQRF programmer (e.g. CK-USB-04x). The procedure is similar to uploading a user program.

More plug-ins can be used at the same time.

Then all plugged-in functions are available like standard system ones.

Flash memory area dedicated to user OS plug-ins can be partly or completely used as an extension of user code (program) memory. See chapter *Flash memory* and IQRF IDE 4 Help.

HWP (Hardware Profiles) based on the DPA framwork are also implemented as plug-ins.

OS upgrade

All IQRF transceivers TR-7xD allows **upgrade** to higher OS version possibly released by the IQRF manufacturer. **Downgrade** (down to a lower version) is possible too.

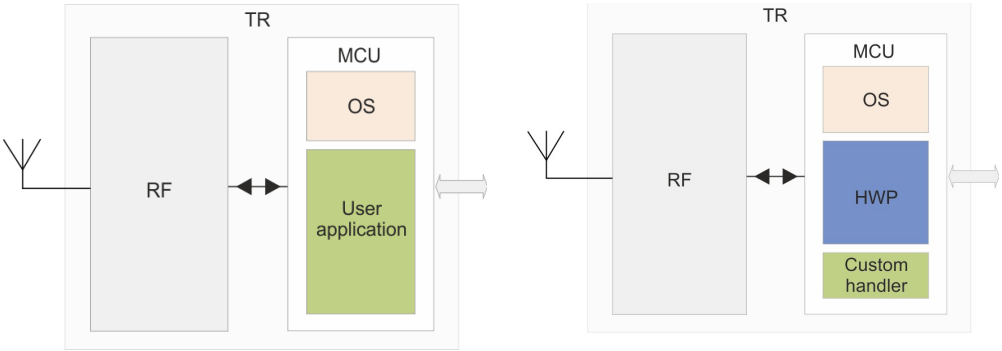
All upgrades and downgrades can be done at the factory or at the user itself, except of the following cases which are possible at the factory only:

- Downgrade from v4.03D
- Downgrade from OS v4.xx down to v3.xx
- Downgrade from v3.09D down to v3.08D or lower

OS	Up/downgradeability at the user
. . .	
4.03D	↑
4.02D	↕
4.00D	↑
3.09D	↑
3.08D	↕
3.07D	↑

DPA

IQRF TR transceivers supports two approaches:
- Programming under IQRF OS
- Programming under IQRF DPA (data controlled)



DPA (Direct Peripheral Access) is an optional higher communication layer above IQRF OS. It is an optional open protocol and framework enabling control by sending commands and receiving responses via standard wired interface (UART or SPI) or via RF. DPA is implemented as a ready-to-use software plug-in called **HWP** (Hardware Profile) to be uploaded into the transceiver. Then no programming is needed. However, it is possible to modify the existing HWP functionality by a user-specific **Custom DPA handler** programmed in C.

Refer to *IQRF DPA Framework Technical guide*.

All IQRF transceivers support both OS and DPA approaches from IQRF OS v4.02D.

RF parameters

- **RF bands**:
  - **868 MHz** or **916 MHz**, configurable by *TR Configuration* in IQRF IDE. Default is 868 MHz, default 916 MHz is available on request.
  - **433 MHz**, implemented in different hardware version of TR transceiver (TR-7xD-433) with fixed band. See the datasheet of respective TR transceiver [7].
- **RF channels:** Within a given band, there are separate frequency channels. RF communication at one of these channels does not interfere with communication at other channels.
  - **Operating channel:** A channel for common RF communication. It is stored in TR Configuration and used as default after TR power on. The Configuration can be changed by *TR Configuration* in IQRF IDE. In networks, the operating channel is inherited from the network Coordinator and stored in Node's Configuration during inclusion the Node into the network (bonding). Operating channel can be dynamically changed during the operation.
  - **Service channels:** Bonding and Smart connect (see chapter *Bonding*) run on fixed 3 channels dedicated for this purpose.

  RF channels and all related parameters depend on TR type. See Appendix *Channel maps*.
- **Bit rate:** 19.8 kb/s.
- **RF output power:** up to 18 mW, SW selectable in up to 8 steps.

Users have to ensure observing local provisions and restrictions relating to the use of short range devices by software, e.g. the CEPT ERC/REC 70-03 Recommendation and subsequent amendments in EU.

The transmitter must operate at the RF **band** with respect to local regulations, e.g. 868 MHz or 433 MHz in Europe. Channel selecting is up to the user application. However, some additional restrictions must be observed in some cases regarding allowed channels and RF output power (with respect to the antenna used) due to the local provisions.

Examples:

- TRs used with high gain antennas must transmit with reduced output power.
- TRs to be operated in Israel are limited to 916 MHz band and channels from 133 to 140 only. This is restricted from the factory for the TR-7xD-IL transceiver versions.
- TR-77Dx transceivers used for fire detectors and similar applications must operate in channels 45 to 67 only and with RF power limited up to level 6. This must be ensured by the user SW application.

See *Appendix 2 – Channel maps* for details.

Microcontroller

IQRF OS for TR-7xDx is implemented in the **PIC16LF1938** MCU (8-bit microcontroller by Microchip) – datasheet see [8].

PIC hardware resources and their utilization in TR modules with OS:

PIC HW resources		Utilization
Program memory	Flash	4544 instructions, split into 2 areas. See chapter Program memory (Flash).
Data memory	RAM	96 B – user data 256 B – communication/system buffers
	MCU EEPROM	Node: 160 B user data + 32 B application data
		Coordinator: 32 B user data + 32 B application data
	Serial EEPROM	16 KB user available
I/O pins	TR-72D	6 × I/O, 2 × LED
	TR-75D	9 × I/O, 1 input only
	TR-76D, TR-77D	11 × I/O, 1 input only
	TR-78D	4 × I/O, 2 × LED
A/D converter		10 b. Multiplexed S&H. Number of pins depends on TR type.
D/A converter		5 b. See Application examples [10]. Allowed range is from VSS to VDD only. Available on transceivers with not shared MCU pins on the Cx SIM pads, e.g. TR-76D. Not available e.g. in TR-72D.
Analog comparator		See Application examples [10]. Allowed CIN+ input is GND only.
Voltage reference (FVR)		Dedicated to OS. Do not change.
Serial communication	SPI (slave)	Supported by OS in background
	I^2C	Implemented by PIC HW module and user function – see Application examples [10]
	UART	Implemented by PIC HW module and user function – see Application examples [10]
PWM output		Implemented by PIC HW module – see Application examples [10]
Interrupt		User available. Global interrupt can be disabled just for a short period if necessary.
Stack (for subroutines)		Max. 5 levels of subroutine calling is allowed.
Timer6		Fully user available.
Power-on reset		Utilizes HW filter to eliminate improper power-up rising and spikes to some extent.
Brown-out reset		Can be enabled and disabled by SW. Default disabled.
Power-up timer		Enabled
Watchdog		Default disabled, can be enabled by SW (`SWDTEN = 1`). Time-out can be set from 1 ms to 256 s, default ~ 4 s. WDT time-out varies with temperature, supply voltage and other conditions from part to part. See PIC datasheet [8].
Oscillator		Internal RC, 16 MHz (250 ns instruction). Do not switch to another clock. IQMESH timing is not derived from this but is calibrated from RF IC with crystal precision.
Configuration words ("Fuses")		`CONFIG1 = 0x2A0C, CONFIG2 = 0x1EFF`

These resources partly depend on TR type and can be under OS supervision and the user should access them in accordance with this manual and possible requirements resulting from hardware construction of the module and OS implementation.

Configuration changes and direct access to some resources by the user can be limited or not allowed at all. Serviceability of some resources depends on using of some other ones at the same time (some hardware communication modules, pins and memory areas are shared for more functions, …).

Parts of memories are dedicated to PIC core, peripherals and operating system. Direct access (via the `EECONx` registers) to the EEPROM and Flash is not allowed at all, extra OS functions are intended for access to EEPROM. Flash memory is user accessible for uploading the program and OS plug-ins to the microcontroller using the IQRF development kits only. Indirect RAM access using the `INDFx` registers is allowed read only due to security reasons. IQRF OS provides complete support for indirect addressing using extra system functions. Not dedicated user inputs/outputs, peripherals (e.g. I^2C and UART) and RAM locations can be accessed directly according to user's needs.

Details see datasheets of the transceiver modules [7], PIC datasheets [8] and Appendix *RAM and EEPROM maps*. In doubt, refer to IQRF support by the manufacturer [6].

MCU pins

Refer to MCU datasheet [8].

MCU pins are mapped in (typically 8-b) ports named `RA`, `RB`, `RC` and so on. Digital pins are bidirectional. There are three important MCU (typically 8-b) registers to control each port: `TRIS`, `PORT` and `LAT`. Every pin is configured by corresponding bit in these registers:

- **TRIS** Selects port direction. E.g. `TRISA.0 = 0;` configures the `RA0` pin as output and `TRISA.0 = 1;` configures this pin as input.
- **PORT** Register to read the level on the pin. E.g. `if (PORTA.0 == 1) …` tests the level of pin `RA0`.
- **LAT** Register to write the level on the pin. E.g. `LATA.0 = 1;` writes log. 1 to pin `RA0`.

There are predefined declarations for every pin in `IQRF-memory.h` header file. Example:

```
#define _C1_IN   PORTA.0      // C1 as input
#define _C1_OUT  LATA.0       // C1 as output
#define _C1_TRIS TRISA.0      // C1 direction
```

Caution

Whenever you switch an MCU pin as output (using the `TRISx` control register), you should set the required log. level via the `LATx` control register (but not the `PORTx` control register) **just before** (to avoid possible collision with OS on the MCU I/O port).

Refer to the MCU datasheet [8], the *read-modify-write* feature.

Example

```
LATA.0 = 1;        // Preset the output value
TRISA.0 = 0;       // Configure the pin as output
if (PORTA.1 == 1)  // Read the input value
  ...              // Do something
```

The dedicated MCU pin for **interrupt on change** at TR transceivers is RB4. It is connected to TR pin C5 (for TRs for SIM mounting, e.g. TR-72D) or Q12 (for TRs for SMT mounting, e.g.TR-76D).

IQRF development tools (e.g. CK-USB-04A and DK-EVAL-04A) with a TR module for SIM mounting, e.g. TR-72D (but not with a TR module for SMT mounting, e.g. TR-76D) use this RB4 pin to connect the User pushbutton (SW1), active low.

See respective PIC datasheet [6] and IQRF OS User's guide [1], chapters *MCU pins* and *Interrupt*.

Memories

For memory purposes, the IQRF OS uses internal memories of the microcontroller (Flash, RAM and EEPROM) and an on-board serial EEPROM.

For short, EEPROM inside MCU is called EEPROM, while serial EEPROM is called EEEPROM in this document.

Individual parts of memories are:
- Dedicated to the MCU core and MCU peripherals
- Dedicated to the OS
- Other areas are available for the user

Memories can be under OS supervision and the user should access them in accordance with this manual and possible requirements resulting from hardware construction of the TR module, MCU architecture and OS implementation. Additionally, when DPA is used, some memory parts dedicated to DPA must be accessed in accordance with the *DPA Framework Technical guide* only.

Illegal modification of dedicated memory locations can cause the system malfunction.

Several header files (with the `.H` extension) are included in installation package of IQRF IDE development environment. They are intended for C compiler to provide linking the OS with the user program. Of course, these text files could also serve to user‘s survey concerning memories – but the user should nowise modify them. (The `16F1938.h` is based on standard file made by the C compiler manufacturer that is why it contain some IQRF irrelevant information.)

User’s own definitions should be placed to extra user header files. Names of user variables must not collide with the names predefined in delivered header files.

Refer to Appendix 1 – Memory *maps*.

Before standard TR operation, the content of the Flash and both EEPROMs can be uploaded into the TR in programming mode in a standard way using IQRF development kit (programmer) CK-USB-04A with IQRF IDE servicing program [9]. For advanced engineers, it is also possible to develop one's own upload software and upload TRs without CK-USB-04A. Refer to the *IQRF SPI Technical guide* or *IQRF CDC Technical guide*, chapters *Upload* in this case.

Memory areas from the application point of view:

- **OS**

  IQRF operating system.
- **OS plug-in**

  A special executable code provided by IQRF producer in some cases, especially to modify IQRF OS (to extend OS functionality, upgrade OS version, fix a bug etc.). None, one or more plug-ins can be uploaded at the same time.
- **TR configuration**

  TR configuration means an obligatory predefined basic TR setup, see chapter *TR configuration* below. It must be uploaded into EEPROM and Flash (and can be read) in programming mode only.
- **Application code**
  - Either an application **program** created by the user’s application engineer
  - Or alternatively a ready-to-use **HWP** (DPA plug-in) provided by IQRF producer (with optional **Custom DPA handler** created by the user’s application engineer or proveded by IQRF producer).
- **Application data**

  May optionally be located in Flash and/or EEPROM and/or EEEPROM. Unlike the Flash data, data uploaded into EEPROM and EEEPROM can be changed later on during normal TR operation.

Depending on memory type and purpose, the data to be uploaded can be stored in files with the following extensions:

.HEX

All application contents (for all TR memories) specified / created by the user except of TR configurations can be included in the source code in C (but EEEPROM up to the first 2 KB only) and exported to a .HEX file generated by the C compiler in standard Intel HEX format INHX8M. For description, see *CC5X C compiler User's guide* (included in IQRF IDE) or *IQRF DPA Framework Technical guide*, chapter *Custom DPA Handler Code at .hex File*.

Caution: Addresses in .HEX file are multiplied by two.

.IQRF

HWPs and OS plug-ins are provided by IQRF producer as ready-to-use files in proprietary encrypted .IQRF format.

.TRCFNG

A proprietary binary .TRCFNG format is used to export and subsequently import TR configuration. See *IQRF SPI Technical guide* or *IQRF CDC Technical guide*, chapter *TR upload* for detailed format if needed.

Access key and *User key* are not exported / imported due to security.

See *IQRF IDE Help*, chapter *TR configuration window* how to create the .TRCFNG file.

Program memory (Flash)

It is primarily intended for **executable code**. That is why it is organized in **14 b words** optimized for machine instructions. However, the Flash memory can also store **data**, typically in arrays, e.g.

```
const char byte_table[] = {0x57,0x04,0x0E};
```

The content remains stored there even after power off. Overwriting is not unlimited, number of erase/write cycles is about 10 000 typically.

Access to Flash via registers `INDFx` or `EECONx` (see MCU datasheet [8]) is not allowed for security.

16 Kwords of the total Flash memory is organized in separate areas dedicated to IQRF OS, TR configuration and user application. The application area consists of two parts – Standard and Extended. See *Appendix 1 – Flash map*.

- **Standard Flash memory**

  Area `0x3A00` – `0x3FFF` (1536 machine instructions) is available for application program.

- **Extended Flash memory**

  Area `0x2C00` – `0x37BF` (3008 machine instructions) is a part of locations dedicated to plug-ins and can also be used (partly or completely) as additional space for application program. It contains two codepages. The CC5X compiler can not address them automatically, the `#pragma` directives must be used for it. Therefore the code must be devided by the user.

  Example:

```
void APPLICATION()              // Code in standard Flash
{
  while(1)
  {
    testLEDR();
    testLEDG();
  }
}

#pragma origin __EXTENDED_FLASH
void testLEDR(void)             // Code in first page of extended Flash
{
  pulseLEDR();
}

#pragma origin __EXTENDED_FLASH_NEXT_PAGE
void testLEDG(void)             // Code in second page of extended Flash
{
  pulseLEDG();
}
```

  See the `E15-EXTENDED_FLASH` example.

- **TR configuration memory**
- **OS memory**

  Remaining areas are dedicated to OS and OS plug-ins.

User program should begin from address `0x3A00`. It is automatically arranged by the IQRF header files.
`void APPLICATION()` must be the first function in user code for this. See IQRF code examples.

Tip: To place a code at a specific location in Flash, the `#pragma origin ADDRESS` directive for C compiler is intended.

Data memory (RAM)

RAM data is fully under supervision of running program and is lost after power off.

RAM access implemented in MCU

Direct RAM access

The MCU logical RAM space is devided to 32 banks (0 – 31, up to 128 B each) with the same architecture:

- Core: dedicated to MCU core, shared (12 identical registers) in all 32 banks
- SFR: Special function registers: dedicated to MCU peripherals
- GPR: General purpose registers
- Common: like GPR but shared (16 identical registers) in all 32 banks

The Core and Common registers are directly accessible without bank preselection. But to access individual SFR or GPR registers directly (e.g. `userVariable = 0`), just the address offset (see the table) is used and given bank must be preselected first by the compiler. See the table of one bank on the right and PIC datasheet [8] for details.

Address offset [hex]	Memory region
00 0B	Core
0C 1F	SFR
20 6F	GPR
70 7F	Common

Indirect RAM access

MCU allows to access GPRs also indirectly in a non-banked methods using addressing via FSRx register pairs. There are two possibilities:

- **Traditional access** using the absolute addresses containing the bank as well as the address offset (FSRx = `0x000` to `0xFFF`). It is available also for the Core, SFR and Common registers. GPR locations available by OS for the user are FSRx = `0x5C0` to `0x5EF` and FSRx = `0x620` to `0x64F` (see below).
- **Linear memory:** Besides of absolute addresses, GPR registers can also be accessed by virtual addresses (FSR = `0x2000` to `0x29AF`) completely independent on banking. Among others, this enables linear access to long user arrays. GPR locations available by OS for the user are FSRx = `0x2390` to `0x23EF` (see below).

RAM architecture implemented in TR modules by OS

- RAM dedicated to **OS**:
  - IQRF **communication** (RF and SPI) is packet oriented therefore buffer servicing is supported. Specialized OS functions are available for comfortable buffer clearing and buffer to buffer data copying. There are four basic buffers primarily dedicated to communications and block operations.
    - **`bufferRF`** (`0x4A0` – `0x4DF`), 64 B – for RF communication.
    - **`bufferCOM`** (`0x3A0` – `0x3DF`), 64 B – for serial wired communication (SPI, …).
    - **`bufferINFO`** (`0x320` – `0x35F`), 64 B – for OS and user block operations

    These communication buffers are especially intended for transferred data and not for other user data. Additionally, OS uses them as work buffers in some cases. E.g. `bufferCOM` can be destroeyd in background by SPI master if SPI is not disabled. See also side effects in IQRF OS Reference guide [1].

    Tip: If there is a need to avoid possible `bufferCOM` destroying in background, SPI must not be active that time.
    - **`bufferAUX`** (`0x420 – 0x45F`), 64 B – auxiliary buffer, to store `bufferINFO` temporarily (by function **`swapBufferINFO`**) to make it free for other operations.
  - Array **`networkInfo`** (`0x2A0` – `0x2BF`), 32 B is an area dedicated to network system information, partly accessible to the user in accordance with IQMESH specification.
  - System variables
    - Some of them can be modified by the user (`toutRF`, `userStatus`. …)
    - Some of them are read only (`RFchannel`, …)
    - Others are not documented, the user need not access them
  - OS work variables (not documented, the user need not access them)
- RAM available for the **user**:
  - Areas `0x5C0` – `0x5EF` (48 B) in RAM bank 11 and `0x620 – 0x64F` (48 B) in RAM bank 12. They can be accessible in a single block up to 96 B using linear data memory access ranging `0x2390` to `0x23EF` (see above). Preselection of bank 11 is automatically arranged by OS. All OS functions return with the bank 11 preselected.
  - Additionally, two `userReg` registers (`0x70` and `0x71`) are available in the Common area shared for all banks.

Mapping of GPR locations available by OS:

Bank	Address		Space
	Linear	Traditional	
11	2390	5C0	user 48 B
	23BF	5EF	
12	23C0	620	user 48 B
	23EF	64F	

See Appendix 1, RAM map.

Tip: To minimize code length due to bank switching by the compiler implied from access to user RAM, it is recommended:

- Map all individual variables (needed to be accessed in random way) to bank 11.
  Default mapping of user variables starts from the beginning of user part of bank 11 and bank 12 is used not until bank 11 is full, unless otherwise stated.
- Map possible arrays, user buffers etc. to bank 12. Example:

```
uns8 i, j;                // This will be mapped in bank 11
#pragma rambank = 12
uns8 xx[37];              // This will be mapped in bank 12
#pragma rambank = 11
uns16 x, y;               // This will be mapped in bank 11
```

- Access bank 12 just by indirect addressing (see below) using dedicated OS functions readFromRAM and copyMemoryBlock and macro writeToRAM.
- Then bank 11 is always kept as the current one in user program. It is automatically arranged by OS (bank 11 is selected after OS boot as well as after finishing of every OS function). Thus, in such arrangement the compiler does not need to switch banks at all.

Indirect addressing of variables

- Due to security reasons, indirect writing to RAM using registers `INDFx` is denied (for either traditional or linear addressing). Thus, arrays are not allowed to be addressed by variable indexes when written. (A[1]=0 is allowed, A[i]=0 is restricted due to writing to `INDFx` by the C compiler.) No such restriction is applied for reading (X=A[i] is allowed).
  IQRF IDE issues a warning and OS performs the `reset` instruction in case of such denied access. See IQRF OS Reference Guide [1].
- Instead of this, IQRF OS allows to write to indirectly addressed RAM using dedicated system functions `setINDF0` and `setINDF1`. Operation is the same like writing to `INDF0` and `INDF1` but without the restriction above. Another possibility is using OS function `readFromRAM` or macro `writeToRAM`. See IQRF OS Reference Guide [1] and example E06–RAM [10].

Note
All IQRF OS functions except of `setINDFx` can modify `FSRx` registers.

RAM access restrictions

In addition to `INDF0`, `INDF1` accessible read only and `EECON1` and `EECON2` inaccessible at all, the following MCU registers can be directly read but are protected against direct writing for security reasons:

- Writing to registers `TMR0`, `OPTION_REG`, `CMOUT`, `SSPCON1`, `CCP1AS`, `CCP3AS`, `IOCBN` and `TMR4` is allowed by the `writeToRAM()` macro.
- Writing to registers `TMR1L`, `PCON`, `BORCON`, `SSPCON2`, `PSTR1CON`, `PSTR3CON`, `IOCBF` and `PR4` is allowed by the `writeToRAM()` macro or by bit-oriented machine instructions (e.g. `IOCBF.4 = 0`).

For several more frequent registers (e.g. to control BOR and wake-up on pin change) there are macros to make this easier. See the IQRF OS Reference guide [1], chapter *Macros*.

When attempted to access locations in an illegal way, restricted instructions are replaced by the reset instruction.

Tip: Such type of MCU reset can be identified by the `_RI` flag in the `userReg0` register (see chapter Reset).

Caution
RAM dedicated to MCU core, MCU peripherals or OS should be accessed by the user in accordance with MCU datasheet, requirements resulting from hardware construction of the TR module, OS implementation and this manual. Illegal modification of dedicated memory locations can cause system malfunction.

Data memory (EEPROM inside the MCU)

Individual parts of internal EEPROM:

- **User data:** 160 B for peer-to-peer or IQMESH Node devices (from `0x00` to `0x9F`). No EEPROM user data is available for IQMESH Coordinator. This area is default cleared (filled with `0x00` values) from the factory.
- **Application info:** It is a 32 B block in EEPROM inside the MCU (from `0xA0` to `0xBF`) dedicated to the user application (for Nodes as well as Coordinators, default cleared from the factory). The user can use this area for any information concerning user application such as configuration, identification and similar purposes. It is especially useful for repeatedly employed (often permanent) data.

  It is possible to read this data directly to the `bufferINFO` by macro `appINFO()` and possibly compare it with the data just received via RF (by the `compareBufferINFO2RF()` function).
- **Dedicated to OS:**
  Remaining EEPROM area (`0x00` – `0x9F` for the Coordinator and `0xC0` – `0xFF` for all devices) are dedicated to OS. It is not accessible by the user. Refer to Appendix *Coordinator bonding and discovery data* for description.

EEPROM access:

- Initial values can be specified in application (source) program to be written to EEPROM while the program is uploaded into the microcontroller. Virtual address range `0xF000-0xF0FF` must be used instead of physical range `0x00-0xFF` when using `cdata` and similar C statements (e.g. `__EEAPPINFO = 0xF0A0`) to address EEPROM. See example E01-TX and E04-EEPROM.
- The microcontroller can read/write data from/to EEPROM under user program control while the application is running using general OS functions for accessing EEPROM (`eeWriteByte`, ...). Physical addresses (`0x00`–`0xFF`) are used in this case. Access via `EECONx` (see MCU datasheet [8]) registers is restricted due to security. See IQRF OS Reference Guide [1].

EEPROM data remains stored even after power off. Overwriting is not unlimited, number of erase/write cycles is 1 000 000 min., (typically 10 000 000). EEPROM is especially intended for configuration parameters and data. The user must avoid exceeding the number of erase/write cycles allowed. Note that some other OS functions (bonding, discovery etc.) write to EEPROM as well.

Data memory (serial EEPROM)

External optional 256 kb EEPROM with serial I2C interface. The upper 16 KB (from `0x4000` to `0x7FFF`) is dedicated to OS and is not available for user applicaton. The lower 16 KB (from `0x0000` to `0x3FFF`) is available for user application. This range may be a subject to change in future IQRF OS versions based on added functionality.

Refer to Appendix *Coordinator bonding and discovery data* for description.

Serial EEPROM access:

- The lowest 2 KB are accessible by the *pragma* directive: Initial values can be specified in application (source) program to be written to serial EEPROM while the program is uploaded into the microcontroller. Serial EEPROM accessible virtual address range is `0x0200– 0x9FF`. Virtual addresses are intended for this case (initializing using the *pragma* directive) in programming mode only, not by the `eeeWriteData` and `eeeReadData` functions (see below) during normal TR operation. Values should be specified in bytes (not in two-byte constants etc.). Refer to example E04-EEPROM.
  Example to initialize serial EEPROM by two `0xFF` values:

```
#pragma cdata[__EEESTART] = 0xFFFF          // Incorrect
#pragma cdata[__EEESTART] = 0xFF, 0xFF      // Correct, stored from address 0x000
```

  By the `#pragma` directive, the lowest 2 KB memory locations can be initialized with specified data. But complete 64 B memory blocks are initialized. Not specified locations in these blocks are cleared.

  Example: `#pragma cdata[__EEESTART + 0x220] = 0x01, 0x02, 0x03` results in:

Location (hex)	0 ... ... 1FF	200 ... 21F	220	221	222	223 ... 23F	240 ... ... 7FF
Value	Unchanged	0	1	2	3	0	Unchanged

- By OS functions: Data in the application is accessible for read and write using OS functions `eeeWriteData` and `eeeReadData` in 1 B to 64 B data blocks, regardless of memory block boundaries.
  Refer to the IQRF OS Reference guide [1] and example E04-EEPROM.

Serial EEPROM is not cleared (not filled with `0x00` values) from the factory.

Data remains stored even after power off. Overwriting is not unlimited, number of erase/write cycles is 1 000 000 typically. The user should avoid exceeding the number of erase/write cycles allowed. Note that also some other OS functions (bonding, discovery etc.) write to serial EEPROM as well.

The presence and proper functionality of serial EEPROM can be checked:

- In IQRF IDE by TR configuration (see below).
- In application software by checking the `eeeReadData` or `eeeWriteData` return value or the `_eeeError` flag. This flag is set by OS whenever a problem in communication with serial EEPROM occurs and stays set until being cleared by the user.

To reduce power consumption, it is possible to disconnect power supply from the temperature sensor and serial EEPROM (both at the same time) and reconnect it only when temperature is measured or serial EEPROM is accessed. Macros `eEEPROM_TempSensorOn()` and `eEEPROM_TempSensorOff()` are intended for it. See IQRF OS Reference guide [1], chapter *Macros*. IQRF OS disconnects these peripherals when entering Sleep or Deep sleep mode and reconnects it after wake-up. This feature must be used with respect to OS which uses serial EEPROM to store some networking information, e.g. during Discovery. That is why it is recommended to utilize such power management for non-networking applications only.

Module identification

MID

Every IQRF transceiver has a unique identification 4 B number `MID` (Module ID) assigned from the factory. It is fixed and not intended to be changed. `MID` and other information about the transceiver and OS are accessible via the `moduleInfo()` function which stores this data to the `bufferINFO`. See the IQRF OS Reference guide [1], `moduleInfo` for Module data format and other details.

Module identification can be displayed by the IQRF IDE development environment.

IBK

Every IQRF transceiver has a unique 16 B number `IBK` (Individual Bonding Key) assigned from the factory. It is fixed and intended for user-friendly including the TR into the network (bonding). See chapter *Smart connect*.

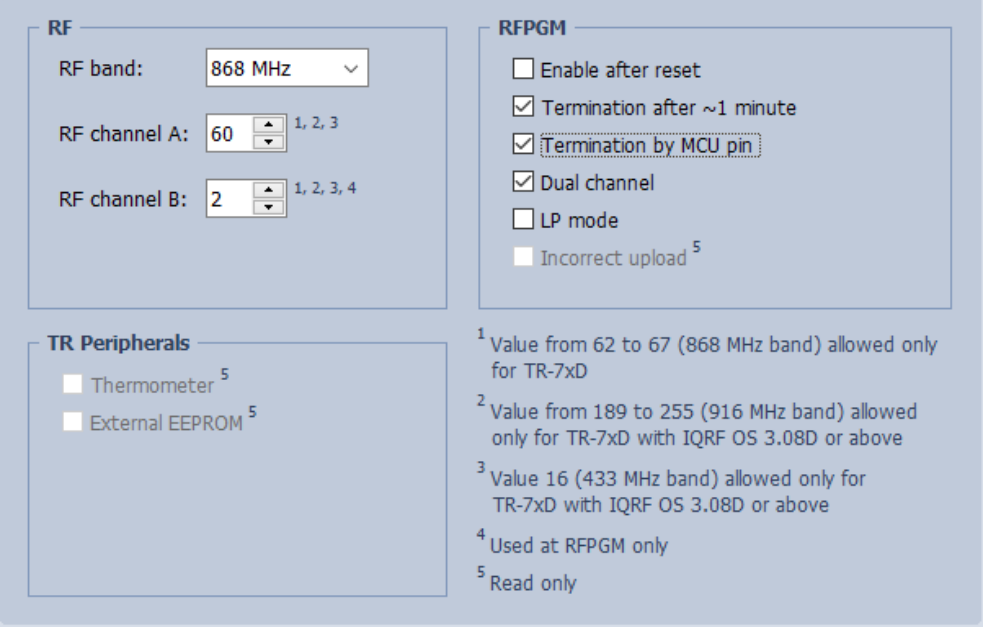
TR configuration

TR configuration is intended to preset default OS features and parameters influencing fundamental behavior of TR module after reset. The configuration is accomplished in IQRF IDE (menu Programming → TR Configuration). The configuration can be uploaded into TR module (and partly also wirelessly via RFPGM), read out from the TR module (except of encryption passwords and keys) and stored as a part of IQRF IDE Project. There are three configuration categories: OS, HWP and Security.
It is also possible to export / import the configuration to / from a `.TRCFNG` file. See IQRF IDE Help for details.

OS

RF

- **RF Band**
  RF band selection is applicable just for TR modules supporting more bands (868 MHz / 916 MHz) and by wired upload only (not by wireless RFPGM upload).
- **RF Channel A**
  RF channel A selection. It is used for common RF communication and as the first (or the only) channel for RFPGM wireless upload.
- **RF Channel B**
  RF channel B selection. It is used as the second channel for RFPGM wireless upload only and applicable just if Dual Channel is activated.



RFPGM

This selection corresponds to OS function `setupRFPGM`. See IQRF OS Reference guide. It is applicable for wired upload only (not by wireless RFPGM upload).

- **Enable after reset**
  Activates RFPGM invoking by TR module reset.
- **Termination after ~1 minute**
  Terminates RFPGM automatically ~1 minute after reset.
- **Termination by MCU pin**
  Terminates RFPGM by dedicated pin of the TR module.
- **Dual Channel**
  Single or dual channel selection.
- **LP mode**
  RFPGM receiving mode setting. If set, then LP (low power) Rx mode is used during RFPGM. Otherwise STD (standard) Rx mode is used.
- **Incorrect upload** (Read only)
  - Unchecked: indicates that the last RFPGM has successfully been completed.
  - Checked: Incorrect upload. Then (even after subsequent reset) the TR module stays in RFPGM mode (the application is not launched).

  This indication is cleared by:
  - Correct upload, either wired or wireless (RFPGM)
  - TR Configuration read / write
  - Entering to (wired) programming mode

Refer to Appendix *RFPGM* for details.

TR Peripherals

- **Thermometer** (Read only)
  Indicates the presence of temperature sensor on the TR module.
- **External EEPROM** (Read only)
  Indicates the presence of external EEPROM on the TR module.

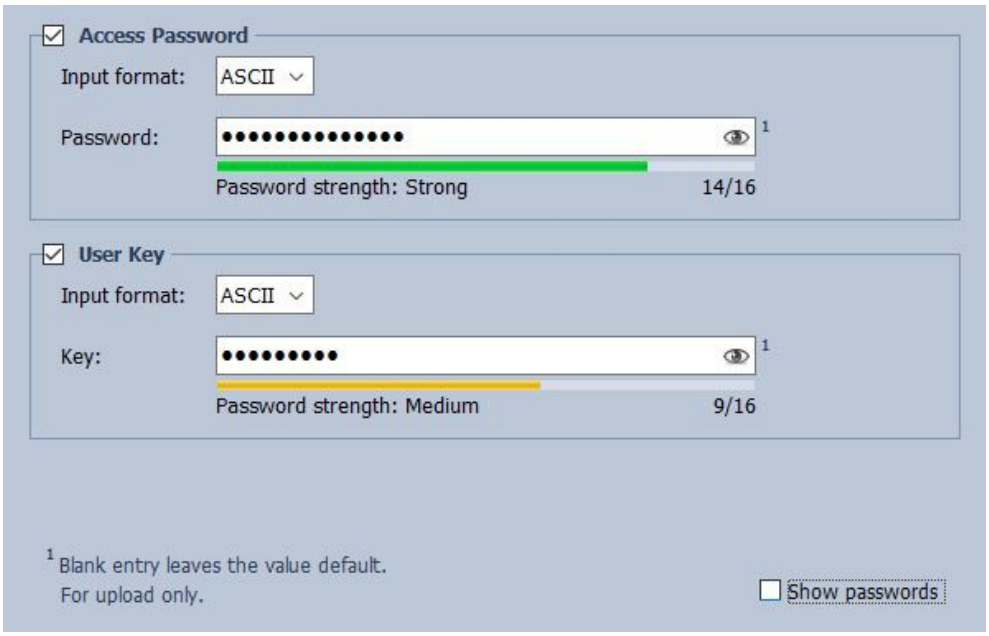
Refer to IQRF IDE Help for details.

HWP

Refer to IQRF DPA Framework Technical guide.

Security

Access Password



- Only for applications using networking communication.
- Intended to store Access password (0 - 16 characters) to TR transceiver. If less then 16 characters is entered, blank (number 0x00) is used for all omitted characters. If the field is left completely empty (no characters filled), the default password (16x number 0x00) is used (the same as the default set from the factory.
- This password is used for:
  - Bonding
    To be bonded as a Node to the network (except of the Smart connect), the TR must have the same Access password as the Coordinator. Leaving the default factory password (see above) unchanged is possible but not recommended. Every network should use a different password.
  - Access via DPA Service Mode. See IQRF DPA Framework Technical guide.
  - DPA device restoring. See chapter *Transceiver replacement*.

Access password can be entered in ASCII or HEX format.

Access password can also be changed in application SW by function `setAccessPassword(x)`.

User Key

- The key for User encryption. See chapter *User encryption*.
- Enter a value up to 16 characters. If less then 16 characters is entered, blank (number 0x00) is used for all omitted characters. If the field is left completely empty (no characters filled), the default key (16x number 0x00) is used (the same as the default set from the factory.
- The User key can be entered in ASCII or HEX format.

Control

Operation modes

The TR modules can work in four modes:

- **Programming:** The user program, OS plug-in and/or TR configuration can be uploaded to the TR (including EEPROM content). This mode is available using the appropriate IQRF development kit and IQRF IDE development environment. See the IQRF Quick start guide [11].
- **RFPGM:** The same content can also be uploaded into the TR wirelessly. In this way, more TRs can be uploaded at the same time. See Appendix 3 – RFPGM RF Programming ™.
- **Run:** The TR module executes operation programmed by the user.
- **Debug:** Execution is stopped and data can be downloaded from the microcontroller and displayed by the IQRF IDE. Then the execution can continue.This mode is fully under control of user program and interactive handling in IQRF IDE.

Real time

OS provides an efficient support for real time applications. It has a generator of time intervals running on background and appropriate functions. Basic period (elementary OS time interval for timing on background – a "**tick**") is 10 ms. Specified number of ticks serve for timing of appropriate processes (delays, LED blinking, communication timeout checks, …) and also enables to create a user timebase. Ticks are derived from internal RC oscillator if RF IC is off (e.g. after `setRFsleep()`) and from crystal otherwise (e.g. after `setRFready()`).

Tip: Use `setRFready()` function if you need precise ticks even when you do not use RF right now.

- The Capture is another efficient timing tool. It is an independent resettable timer (16-bit counter of ticks) freely running on background. It is suitable especially for working with long periods (up to 655 s).
- OS also provides functions for waiting on foreground.
- Short time intervals for timing on foreground can be derived also from instruction timing. The MCU is clocked with internal 16 MHz RC oscillator. Thus, instruction cycle is 500 ns (1 µs for some instructions) – see PIC datasheets [8].

Some OS functions (especially `RFRXpacket` and several delays) share the same internal timers that is why these functions should not be used at the same time. Refer to the IQRF OS Reference guide [1], side effects.

Note that time precision of TR modules depends on precision of internal RC oscillator – see PIC datasheets [8]. Microcontrollers are individually calibrated by the manufacturer but despite of this fact the precision and stability are less than for a crystal oscillators. The precision is sufficient for asynchronous communication (UART) with reasonable speed. RF timing is derived from RF IC with quartz precision through automatic calibration accomplished regularly by OS (except of the time when RF IC is sleeping).

Timer6

Timer6 is a HW peripheral of MCU intended as a programmable timer. It is fully user available. It works in background and can also generate an optional MCU interrupt. It cannot be operated while the MCU is in Sleep mode. Timer6 is not automatically stopped in debug. Refer to the PIC datasheet, chapter *Timer2/4/6 modules* [8] for handling.

Watchdog

To increase the reliability, OS allows to use hardware watchdog of the MCU to recover the system from unexpected events. It is a continuously running independent timer with a programmable overflow period. It should be used that never overflow during correct operation. It is accomplished via the `clrwdt()` instruction always executed in time, i.e. before the watchdog overflows. (This function is implemented not in OS but it is the PIC machine instruction supported by the C compiler). If an overflow occurs it is regarded as a program execution failure and the microcontroller responds with reset. If the failure is not a permanent one, it can lead to system recovering. It is up to the user to place `clrwdt()` at proper location(s). The watchdog can run even in the Sleep mode (see below). Overflow in Sleep results in wake-up and continuing execution but not in the PIC reset.

The watchdog is default disabled and can be enabled by SW via the `SWDTEN` bit (0 – disabled, 1 – enabled). Overflow period is user selectable (even while the program is running) from 1 ms to 256 s by the `WDTPSx` bits in the `WDTCON` register – see PIC datasheet [8]. For easier watchdog handling, macros `setWDTon()`, `setWDToff()` and `setWDTon_xxxx` are available. Refer to the IQRF OS Reference guide [1], chapter *Macros*.

Power down modes

TR module can be put in power down Sleep or Deep sleep mode. Then almost no operation is executed but the power consumption is minimized. To reduce consumption while TR not sleeping, the RF sleep mode is available.

Sleep mode

The microcontroller and RF IC are put in the Sleep mode and other on-board circuitry is disabled. Switching to/from the Sleep mode is fast and waking up requires no or just a simple reinitialization after wake-up.

Switching to the Sleep mode:

The Sleep mode is initiated in software using the `iqrfSleep()` function in appropriate location in the source program. Then all TR hardware resources controlled by the OS are automatically suspended: activity of the TR module including the RF circuitry, temperature sensor, microcontroller as well as its peripherals (stopping of timers, disconnecting of internal pull-ups, ...).

Before switching to the Sleep mode:

- Power consumption should be minimized even for hardware resources of TR controlled by the user:
  - Possible PIC internal peripherals and possible external peripherals used by the application should be switched off.
  - No PIC pins must be left as digital inputs without defined input log. level values. See example E14-CONSUMPTION.
  - SPI must be disabled if there is a possibility of unintentional wake-up from SPI master. It must be done in application program.

  See the TR [7] and PIC [8] datasheets.
- The microcontroller should be configured for subsequent wake-up on pin change (if required):
  - Wake-up on pin change is under user's control, default disabled. Either one or both rising or falling edge can be used.
  - To enable, the following sequence is required:

```
GIE = 0;                              // Global interrupt disabled
writeToRAM(&IOCBN, IOCBN | 0x10);     // Negative edge enabled. Instead of IOCBN.4 = 1;
                                      // IOCBN cannot be accessed directly due to OS
                                      // restriction.
IOCBP.4 = 1;                          // Positive edge enabled
IOCIE = 1;                            // Interrupt on change enabled
GIE = 1;                              // Global interrupt enabled
SWDTEN = 0;                           // Watchdog disabled
iqrfSleep();                          // Sleep
GIE = 0;
writeToRAM(&IOCBN, IOCBN & 0xEF);     // Negative edge disabled (IOCBN.4). Instead of
                                      // IOCBN.4=0.
IOCBP.4 = 0;                          // Positive edge disabled
GIE = 1;                              // Global interrupt enabled
```

Similar functionality can be achieved by macro `sleepWOC()`. See IQRF OS Reference guide [1], macro `sleepWOC()`.

IOCBF flag is cleared automatically by OS. Flags IOCBN and IOCBP are not changed within the `iqrfSleep()`. Thus, these flags should be handled by the user. See the code above.

Returning to the operating mode (wake-up)**:**

- After watchdog overflow (if enabled)
- After pin change on some pins (depending on the TR type, typically the C5 pin), when configured as inputs (if enabled)
- After power-off/on

Tip: wake-up types can be identified via the `-TO` and `-PD` status flags (in the MCU `Status` register).

After the wake-up the microcontroller continues with execution the command following the Sleep function.

The user can use Sleep and wake-up without any restriction due to OS, all related microcontroller possibilities can be employed – see PIC datasheets [8].

Tip: sleep period can be setup via the watchdog timeout period.

Typical sleep power consumption see the TR datasheets [7].

Deep sleep mode

This mode is the same as the Sleep mode but RF IC is put in the Shutdown instead of the Sleep mode. This is the state with the lowest possible power consumption. However, unlike the Sleep mode, after waking up from Deep sleep the previous RF IC configuration is lost and must be completely reinitialized in application before any RF functionality is utilized. Therefore, Deep sleep mode is suitable especially in applications where very low power consumption is required and long not operating periods are possible. It is primarily intended for TR-76D and other TRs without internal LDO voltage regulator, supplied directly from a battery.

Switching to the Deep sleep mode:

The Deep sleep mode is initiated in software using the `iqrfDeepSleep()` function in appropriate location in the source program. The preconditions are the same as for Sleep mode.

Returning to the operating mode (wake-up)**:**

Waking up can be invoked in the similar ways like from the Sleep mode. There are two typical cases after wake up:

- TR provides what is required (without RF operation) and falls in Deep sleep again.
- TR should provide RF operation as well. In this case the following steps must be accomplished before:
  - `setRFready()` must be called. This functions, when called after Deep sleep, reinitializes RF IC into the default state like after power on.
  - All RF parameters specified by the user must be restored. These are especially RF channel, TX power and parameters specified by the `setRFmode()` and `checkRF()` functions.

RF sleep mode

When no RF functionality is required while TR is not sleeping, the RF IC can be switched off by function `setRFsleep()` to minimize power consumption. RF wake-up can be caused by `setRFready()`, `RFTXpacket()`, `RFRXpacket()` or `checkRF()` functions.

Other PIC peripherals

There are PIC HW resources (I^2C, UART, PWM output, A/D and D/A converters, analog comparator, …) user accessible but not supported by the OS. Refer to datasheet of given TR module [7], the microcontroller [8] and application examples [10].

Due to pin sharing with more peripherals possible conflicts must be avoided by the user. E.g. TR-76D D/A converter must not be used at the same time with OS functions utilizing red LED.

Reset

Initialization of MCU, OS and application. The following reset types (slightly differing in MCU initialization) are available:

- **Power-on reset (POR)**: Utilizes HW filter to eliminate improper power-up rising and spikes to some extent. A lot of applications need no external reset circuitry.
- **Brown-out reset (BOR):** If power supply falls below 1.9 V for 3 µs (typical values), the reset is invoked. This option is default disabled and can be enabled by user SW. BOR should be disabled in Sleep otherwise additional 7.5 µA is consumed. Direct writing to Brown-out control register `BORCON` is restricted (see above). Thus, `SBOREN = …` is not allowed and macros `setBORon()` and `setBORoff()` should be used instead.
  Tip: If possible problems with unstable power supply may be expected, it is recommended to call `setBORon()` as the first command in the application, otherwise unpredictable behavior of TR module can occur.
- **Watchdog (WDT) reset:** After WDT time-out.
- **Reset instruction:** Reset can also be invoked by SW by the `reset()` command. It is not an OS function but MCU machine instruction accessible as a macro. Refer to IQRF OS Reference guide [1] , chapter *Macros*.
  Example:

```
if (Error == 1)
  reset();
```

- **Stack overflow / underflow reset:** After MCU Stack overflow / underflow.

Determining the cause of a Reset

To identify reset type the following status flags are available in the `userReg0` just after boot, independently on possible preceding wired or wireless (RFPGM) upload:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Status flags	**_STKOVF**	**_STKUNF**	**_RMCLR**	**_TO**	**_PD**	**_RI**	**_POR**	**_BOR**
Power on reset	0	0	1	1	1	1	0	x
Brown-out reset	0	0	1	1	1	1	u	0
WDT reset	u	u	u	0	u	u	u	u
WDT Wake-up from Sleep	u	u	u	0	0	u	u	u
Interrupt Wake-up from Sleep	u	u	u	1	0	u	u	u
Reset instruction executed	u	u	u	u	u	0	u	u
Stack overflow reset	1	u	u	u	u	u	u	u
Stack underflow reset	u	1	u	u	u	u	u	u

u = unchanged, x = undefined

_BOR Brown-out reset flag
- _BOR = 0 after Brown-out reset (must be set in software then)
- _BOR = 1 no Brown-out reset occurred

_POR Power-on reset flag
- _POR = 0 after power-on reset (must be set in software then)
- _POR = 1 no power-on reset occurred

_RI Reset instruction flag
- _RI = 0 After `Reset` instruction executed
- _RI = 1 `Reset` instruction has not been executed

_PD Power-down flag
- _PD = 0 after `iqrfSleep`
- _PD = 1 after power-up or `clrwdt`

_TO Watchdog time-out flag
- _TO = 0 after WDT overflow
- _TO = 1 after power-up, clrwdt or iqrfSleep

_RMCLR –MCLR reset flag
- _RMCLR = 0 after –MCLR reset
- _RMCLR = 1 –MCLR reset has not occured

_STKUNF Stack underflow flag
- _STKUNF = 1 after Stack underflow
- _STKUNF = 0 Stack underflow has not occured

_STKOVF Stack underflow flag
- _STKOVF = 1 after Stack overflow
- _STKOVF = 0 Stack overflow has not occured

To fully utilize all these features, reset flags should be properly handled in SW after respective event occurred. Refer to the PIC datasheet [8], (`STATUS` and `PCON` registers and Reset) for details.

Content of RAM registers after resets is strictly defined (set / cleared / not affected / unknown). It partly depends on reset type. Refer to MCU datasheet [8]. Additionally, OS completely clears the user memory area except of the `userStatus` register (address `0x5AD`) which is cleared after power-on reset and remains unchanged after other reset types.

Tip: `userStatus` can be used to help to debug unexpected resets in user programs:

- Fill this register with different values at suspicious locations to identify where unexpected reset occurs.
- You can also use this register as a counter or timer to reveal the cause of the reset.

Interrupt

MCU offers a sophisticated interrupt system supporting various asynchronous events allowed to interrupt the running program, call specified procedure and then continue from the previous location. In brief, all interrupts have the same handling: it can be enabled / disabled by the flag …E and generates a request for interrupt by the flag …F. Additionaly, all interrupts can be enabled / disabled by the flag GIE (global interrupt enable). In fact, due to various peripherals involved, the interrupt system is slightly more complex. That is why refer to the PIC datasheet [8].

OS and IQMESH use the interrupt system to generate system ticks, communication control etc. Additionally, interrupt can also be used by the user.

Interrupt on MCU pin change is available at TR pin C5 or Q12. See chapter MCU pins.

Disabling interrupt

Interrupt can be disabled by the user (`GIE = 0;`) to gain control over precise timing, e.g. to generate a signal with desired waveform. To avoid OS malfunction, disabling is allowed for very short period only (depending especially on communication in the application).

User interrupt

Proper timing is fundamental for correct OS functionality, that is why the user should use interrupt very considerately and in well reasoned cases only.

Specified user routine can be called whenever an interrupt occurs. The following rules must be observed within it:

- User interrupt must be enabled (`_enableUserInterrupt = 1`) when desired and disabled (`_enableUserInterrupt = 0`) when not desired. Default OS condition is disabled.
- The user interrupt routine must start from address `__USER_INTERRUPT` (predefined in `IQRF-macros.h` header file).
- The user interrupt routine must take less than 50 µs.
- User interrupts must not be called in period shorter then 800 µs.
- MCU flag `SSPIF` must not be used to generate interrupt (e.g. for I2C slave communication).
- IQRF OS variables must not be used.
- IQRF OS functions except of `setINDFx` are not allowed.
- Only global or local static user variables are allowed.

If enabled, user interrupt routine is invoked within any interrupt handling. If the interrupt is caused by OS, the user routine is invoked after all OS interrupt handling is finished. User interrupt is not automatically disabled in debug.

Timing accuracy of user interrupt is limited. The order of magnitude of possible difference is 1 ms typically. But in worst case during communication (especially via RF), the user interrupt can be skipped at all in some period(s). That is why the user interrupt is unsuitable for generating of reliable regular periodic interrupts required in any conditions.

There are more sources able to generate a user interrupt (UART, Timer6 etc.). It is also enabled to generate user interrupt on pin change via the `IOCBF` flag.

User interrupt (`_enableUserInterrupt = 1`) should be enabled sooner than any individual interrupt sources (e.g. `TMR6IE = 1`). Possible disabling should be done in reverse order.

See `E013-INTERRUPT` example [10] .

Example using Timer6:

```
…
_enableUserInterrupt = 1;     // Enable user interrupt
TMR6IE = 1;                   // Enable interrupt from Timer6 – not until user interrupt is enabled
…
…
TMR6IE = 0;                   // Disable interrupt from Timer6 – before user interrupt is disabled
_disableUserInterrupt = 0;    // Disable user interrupt

#pragma origin 0x3F00         // User interrupt routine must start here
#pragma library 0
void userIntRoutine(void)
{
    if (TMR6IF) {             // Ignore all other interrupt causes
      _LEDG = 1;              // LED on when Timer6 overflowed
      TMR6IF = 0;             // Clear to reinitialize Timer6 for next interrupt
    }
}
```

Temperature measurement

For temperature measurement the precise (optional) digital on-board sensor MCP9808 (Microchip) with ~0.0625 ºC resolution and 0.5 ºC accuracy in 12 b data format is used. However, the resolution used by IQRF OS is limited to 0.5 ºC. See `E08-TEMPERATURE` example [10] and IQRF OS Reference guide [1], function `getTemperature`. Temperature sensor is connected to MCU via the I2C bus. Presence of the sensor (declared by the "T" postfix in TR module type string, e.g. TR-72DAT) can be identified by TR configuration (see above) and also in application software.

To reduce power consumption, it is possible to disconnect power supply from the temperature sensor and serial EEPROM (both at the same time) and reconnect it only when temperature is measured or serial EEPROM is accessed. Macros `eEEPROM_TempSensorOn()` and `eEEPROM_TempSensorOff()` are intended for it. See IQRF OS Reference guide [1], chapter *Macros*. IQRF OS disconnects these peripherals when entering Sleep or Deep sleep mode and reconnects it after wake-up. This feature must be used with respect to OS which uses serial EEPROM to store some networking information, e.g. during Discovery. That is why it is recommended to utilize such power management for non-networking applications only.

The temperature conditions at the on-board sensor covered by metallic can shilelding can be different from the ambient environment (affected especially by heat dissipation depended on current power consumption).

Battery check

Supply voltage can be measured by the `getsupplyVoltage()` function. See IQRF OS Reference guide [1].

LED indication

Two on-board LEDs (red and green) are controlled as follows:

- By OS: Some important conditions (e.g. the Programming mode) are automatically indicated by IQRF OS.
- By DPA: Additional LED indication is controlled by the DPA layer. For networking only. Refer to the *IQRF DPA Framework technical guide*, chapter *LED indications* for OS and DPA LED indication overview.
- By the user: LEDs can be controlled by the set of specialized OS functions (e.g. `pulseLEDR()`) running in OS background. Direct handling the appropriate MCU pins (e.g. `_LEDR = 1`) is possible but not necessary. Use the LED control OS functions, e.g. `setLEDR()` or macros, e.g. `toggleLEDR()` whenever it is possible.
  Take into consideration whether the LED levels can be modified in background by an OS function or a DPA command. Such a LED command can also be received wirelessly (DPA request).

SPI

Standard serial 4-wire bus in slave mode running in OS background. See separate document *SPI implementation in IQRF TR modules* [5]. SPI is widely used also by the IQRF IDE.

Debug

The IQRF platform provides user with an efficient debugging tool. To use it, the following configuration should be used: The transceiver module plugged e.g. in the CK-USB-04x development kit connected to PC via USB with the IQRF IDE development environment [9].

Debug is directly supported by the OS with the `debug()` function. This can be included in user program wherever you need to stop program executing and evaluate variables, EEPROM content or RAM registers. After uploading user program into the transceiver module the application is running until the `debug()` function is encountered. Then the program stops, the module is switched to the debug mode and data is downloaded and displayed on the screen.

The module stays in debug mode till the user wishes. Then the application program can continue execution until another `debug()` function is encountered and so on. To identify individual debug breakpoints, the W register can be used. Macro `breakpoint(w)` can advantageously be utilized for this. See IQRF OS Reference guide [1] (chapter *Macros*), IQRF IDE Help, header file `IQRF-Macros.h` defining macros and `E06-RAM` example [10]for details.

When entering debug, the application must not have enabled interrupt from any of user peripherals. Debug must not be used within the user interrupt routine.

Debug utilizes SPI to communicate with TR module. Thus, SPI pins must be configured as follows: C5, C6, C7 as inputs and C8 as output. This is arranged by OS after reset. If changed by the user, this state must be restored before calling the `debug()` function.

Possible user circuitry connected to SPI pins must avoid conflicts otherwise failure or damage can occur.

RF

RFcommunication

OS functions allow powerful and user-friendly control of RF communication. From the user's point of view it means working primarily with memory and buffers (R/W operations with RF communication buffer). IQRF OS automatically provides all needed services including full protocol impementation:

- at transmission level: HW setup, coding for transmission, timeouts, ...
- at packet level: preamble, consistency checking, coding, ...
- at network level: routing, including information about the network and device, filtering, discovery, ...

Supported modes:

- **Peer-to-peer:** Two or more peer-to-peer devices, without a network Coordinator. Packets are available for all devices in range and completely managed by the user program. Number of devices is unlimited. Keep `PIN=0` (see below) in this mode. This is the default mode.
- **IQMESH:** Topology with one Coordinator mastering the network and up to 239 end devices (Nodes). Each Node can additionally work as a router on OS background. DPA is the only recommended way for IQMESH implementation. The IQMESH mode is selected by setting the most significant bit (`_NTWF`) of the PIN register to 1. Nodes must be assigned (bonded) to the Coordinator's network. Peer-to-peer ("non-networking") packets are also allowed in IQMESH ("networking").

Memory locations and registers related either to Peer-to-peer or IQMESH:

`bufferRF[64]`	Buffer for RF routines (data to be sent by `RFTXpacket` or received by `RFRXpacket`), 64B
`PIN`	Packet information. See below.
`RX`	Packet addressee (specify before transmitting)
`TX`	Original packet sender (set by OS during receiving)
`DLEN`	Packet length (number of relevant bytes in `bufferRF`), 0-64 (specify before transmitting, set by OS after receiving)
`toutRF`	Timeout for packet receiving (1-255) in number of 10 ms ticks or for LP and XLP modes in cycles (see below, Low power modes). Default value is 50 (500 ms in STD mode).

IQMESH network and its individual devices can be configured very flexibly. IQMESH as well as peer-to-peer packets can be sent and received depending on setup of respective devices. Individual and broadcast packets (for all network members) are supported and user addressing is allowed. IQMESH protocol has been defined as a light and portable to inexpensive microcontrollers with limited resources. One or two byte addressing is chosen.

Background routing is fully suppported. Each Node can provide background routing service for network packets or can be programmed as a dedicated router. Both Coordinator and Node can be realized by a more complex device, a Gateway, providing an interface between IQMESH and other standards. See below (Filtering).

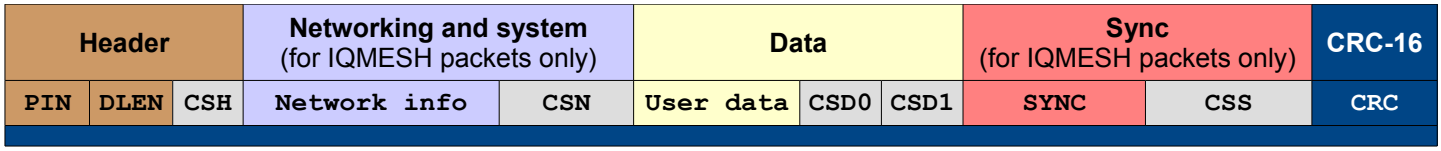
Although IQMESH is very flexible and supports high variability and dynamic changes in configuration (including changes in topology), it is primarily intended for more or less static systems. Devices are included in / excluded from the network by the bonding / unbonding procedure which should be considered to be an installation process by its nature. The Coordinator is not intended to be switched dynamically from device to device in a network. The Coordinator should manage RF communication in the whole network. Nodes are allowed to communicate anytime but it can be recommended just in special cases. In typical applications the Coordinator always initiates any communication. All IQMESH communication is encrypted. See chapter Encryption for details. In addition to "user" packets, IQMESH uses also system packets with auxiliary information (e.g. for bonding, routing etc.). Such system packets are completely transparent from the user's point of view.

Basic network information about current setup of given device (network identification, device number, current network, topology, ...) provides the `getNetworkParams()` function and the `networkINFO` array.

IQMESH packet

IQMESH packet transmission is supported by a lot of additional sophisticated features. The communication is possible even between nodes out of RF range each other – using “hops” via other nodes in range (routing). In addition to the normal operation, every IQMESH device (TR module, gateway, ...) can work also as a router on background. IQMESH can additionally contain specialized plug-and-play routers.

RF packets consists of blocks *Header*, *Networking and System*, *Data* and *Sync*. Every block has its own consistency check (`CSx` checksum). The last part of the packet is IBM CRC-16 checking. The whole networking packet is subsequently encrypted. See chapter Networking encryption.

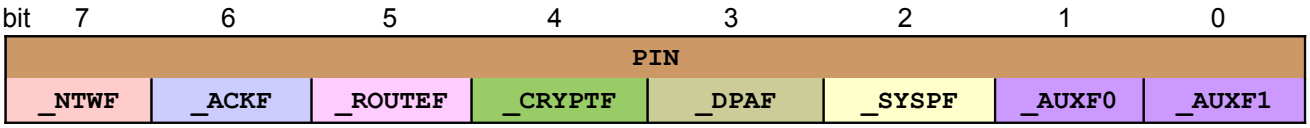


Header

Packet header carries basic information about a packet, such as data length and various flags.

DLEN User data length [in bytes]

PIN Address `0x2A0`



`_NTWF:`

`_NTWF = 0` Peer-to-peer mode

`_NTWF = 1` IQMESH mode (See above, chapter *RF overview*)

`_ACKF:` Acknowledge request. Reserved for future use.

`_ROUTEF`: (For IQMESH only)

`_ROUTEF = 0` Routing not required for outgoing packets.

`_ROUTEF = 1` Routing required for outgoing packets, routing registers `RTHOPS`, `RTTSLOT`, … must be defined.

`_CRYPTF:` Dedicated for OS..

`_DPAF:` Dedicated for DPA and system services.

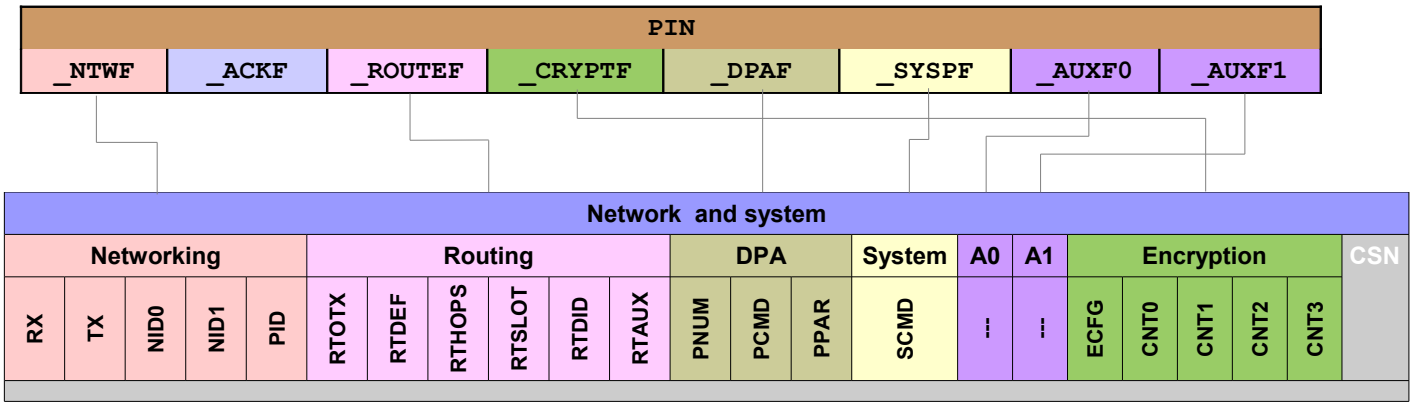
`_SYSPF:` Dedicated for OS..

`_AUXF0:` Dedicated for OS.

`_AUXF1:` Dedicated for OS.

Network and system (applies for IQMESH mode only, i.e. when `_NTWF = 1`)

Networking information block has variable length based on `PIN` flags:



Just five bytes (`RX` to `PID`) are mandatory (present in every IQMESH packet), the others depend on actual situation. For example, Star topology does not need routing information. Setting `_ROUTEF = 0` will make a packet without routing, while after setting `_ROUTEF = 1` six bytes describing the routing are expected to be added to the Network info. This mechanism provides a way to fit various application needs. Network and system info registers are set by the sender and are passed to all recipients via the packet. Thus, the recipient know which hop is actually in question and how long it is to wait before possible forwarding.

`RX` Address of the device the packet is intended to in current network:
- 0 Coordinator
- 1 - 239 Nodes
- 240 - 253 Reserved
- 254 Universal address for prebonded Nodes
- 255 Broadcast

`TX` Address of transmitting sender. It is automatically set by OS by `RFTXpacket()`.

`RTDEF` Routing algorithm. See chapter *Routing algorithms*.

`RTHOPS` Number of hops per packet

`RTTSLOT` Time slot length [in ticks]

`PNUM` Dedicated to DPA (Direct Peripheral Addressing) and system services.

`PCMD` –"–

`PPAR` –"–

Other registers (not documented here) are dedicated to OS and not intended to be used in application SW.

User data

User data from/to the `bufferRF`. The length can vary between 0 and 64 B and is specified in the `DLEN` variable.

Sync

Intended for precise synchronization during routing. For internal OS purpose only.

CRC-16

Standard IBM CRC-16 for the whole packet provides additional security against bit transfer errors.

User interface

Some information about curent system, RF and network parameters is available in the `userInterface` register. Bits marked with “R/W“ can be changed by the user. Bits marked with “R“ are **read only**. Changing of them is not allowed or allowed via respective OS functions only (if applicable).

Address `0x07F`

7	6	5	4	3	2	1	0
R	R	R	R	R	R	R/W	
_networkingMode	_networkTwo	_filterCurrentNetwork	_916MHz	_wasRouted	_wasFRC	_enableUserInterrupt	-

- `_networkingMode`
  - 0: Peer-to-peer (non-networking) topology. Relates to the `setNonetMode` function.
  - 1: IQMESH selected. Relates to the `setCoordinatorMode` and `setNodeMode` functions.
- `_networkTwo`
  - 0: Network 1 selected. Cleared by the setCoordinatorMode function.
  - 1: Network 2 selected. Relates to the `setNodeMode` function.
- `_filterCurrentNetwork`
  - 0: Network filtering disabled. Relates to the setNetworkFilteringOff function.
  - 1: Network filtering enabled. Relates to the `setNetworkFilteringOn` function.
- `_916MHz`
  - 0: RF band 868 MHz selected.
  - 1: RF band 916 MHz selected.
- `_wasRouted` – Corresponds to `wasRouted()` return value.
  - 0: packet was not routed.
  - 1: packet was routed.
- `_wasFRC`
  - 0: FRC packet not received.
  - 1: FRC packet received.
- `_enableUserInterrupt`
  - 0: User interrupt disabled.
  - 1: User interrupt enabled.

Other RF and network parameters

Other parameters are available in RAM registers strarting from `0x5A0`, see Appendix 1 (OS, RF and network parameters). Writing is allowed to the registers marked with R/W, the others are read only. All registers with names starting with `ntw` (addresses `0x5A0` to `0x5AA`) are updated after calling the `getNetworkParams()` function only (e.g. `ntwADDR`).

The `_disabledRouting` flag (`ntwCFG.2`) relates to the `setRoutingOff()` and `setRoutingOn()` functions and is valid after calling the `getNetworkParams` function.

sysReg1

Address `0x5BB`

7	6	5	4	3	2	1	0
	R/W	R/W	R/W	R/W	R/W		
-	_advancedFRCmode	_ignoreForcedRoutingLP	_XLPnoWait	_prebondMode	_systemLEDindication	-	-

The `sysReg1` register contains several control flags intended for complex networking functionality like bonding, discovery and FRC, especially in low power modes. See respective chapters below and IQRF OS Reference guide. Flag `_XLPnoWait` is not implemented in surrent OS version.

sysReg2

Address `0x042`

7	6	5	4	3	2	1	0
				R/W			
-	For internal purpose only		-	_eeeError	-	-	-

The sysReg2 register contains some other control flags. The `_eeeError` flag indicates serial EEPROM access error. It is set by the following functions:

- `eeeReadData` and `eeeWriteData` (read/write error)
- `optimizeHops`
- `discovery`
- `bondNewNode`
- `nodeAuthorization`

See IQRF OS Reference guide.

`_eeeError` clearing is always up to the user.

Other bits in `sysReg2` should not be used by the application.

RF IC modes

RF IC can operate in four modes. Power consumptions mentioned here are just for guidance and relate to RF IC only.

RF Sleep

RF IC is completely switched off (while the MCU can continue running). OS system clock (tick) is derived from MCU internal RC oscillator. This mode is established after reset of the TR module, after `iqrfSleep` or `setRFsleep` and when XLP receiving is finished.

RF Ready

RF IC is switched on and crystal oscillator is running. OS system clock (tick) is derived from from the crystal oscillator. RF receiver is not active (RX chain is off). This mode is established after `setRFready()`, `checkRF(x)`, `RFRXpacket()` or `RFTXpacket()`.

RF Receiving

RF receiver is active (RX chain is on). RF IC works in this mode during `checkRF(x)` or `RFRXpacket()`.
After `RFRXpacket()` or `checkRF(x)` finishing, RF IC is switched to RF Ready.

RF Transmitting

RF transmitter is active. RF IC works in this mode during `RFTXpacket()`. After finishing, RF IC is switched to RF Ready.

Code example illustrating consumption in RF IC modes

Consumptions mentioned here are just for guidance, relate to complete TR module and depend on the TR type.

```
setRFsleep();     // MCU running, RF Sleeping
waitDelay(255);   // about 1.7 mA total

setRFready();    // RF Ready
waitDelay(255);   // about 2.9 mA

toutRF = 255;
RFRXpacket();    // RF Receiving, about 12 mA

RFTXpacket();     // RF transmiting (full RF power), about 18 mA

iqrfSleep;       // MCU Sleeping, RF Sleeping, about 1.7 µA
```

For details and LP and XLP modes refer to `E14-Consumption` example [10].

RF transmitting

It is possible to combine sending peer-to-peer and IQMESH packets (depending on the `_NTWF` flag).

- Peer-to-peer: Prepare data to the `bufferRF`, specify data length (`DLEN = ...`) and simply send the packet via the `RFTXpacket()`. All receivers obtain the data and `DLEN` only.
- IQMESH: To send IQMESH packets, an appropriate setup (Coordinator/Node selection etc.) should be done and the Node should be bonded to a network. Sending itself is similar to Peer-to-peer but the receiver address (and possible routing information) must be specified. Other networking information is added to the packet by OS automatically.

RF packets can be transmitted at RF output power selectable from 8 levels by the `setRFpower(level)` function. Maximal level must be selected with respect to local provisions and restrictions. E.g., in EU, the ERC-REC 70-03 regulation allows radiated power up to 14 dBm (25 mW) only. Therefore, directional high gain antennas such as AN-08 (Yagi) or AN-09 (planar) should be used with properly reduced levels.

Other local provisions and restrictions could lead to output power reduction as well. E.g., see chapter *868 MHz band channel map* for regulation regarding unwanted emissions in EU. For TR-77D series, the highest RFpower (`level = 7`) is not allowed at all.

If bidirectional communication, `PIN` and `DLEN` should be updated before every transmitting followed after any reception.

Packets can be sent in several modes with respect to a receiver power managing mode.

RF packet propagation time

DLEN	Peer-to-peer (PIN = 0x00) [ms]
1	11
10	15
64	38

Times indicated above are for brief guidance only, rounded to ms. For networking packets propagation times see the *IQRF DPA Framework technical guide*, chapter *DPA confirmation*.

Actual time consumed by sending a packet (`RFTXpacket` duration) can be measured on a TR pin by an oscilloscope:

```
…
PIN = 0;
setCoordinatorMode();   // Remove for Peer-to-peer
DLEN = 10;              // Set required length (payload in bytes)
_C1_OUT = 1;            // Rising edge on pin C1
RFTXpacket();           // Send the packet and wait until finished
_C1_OUT = 0;            // Falling edge on pin C1
…
```

See the IQRF OS Reference guide [1] (`RFTXpacket()`) and examples `E01-TX`, `E03-TR` and `E09-LINK` [10].

`PIN`, `RX`, `DLEN`, `RTDEF`, `RTHOPS`, and `RTSLOT` registers must be updated before calling `RFTXpacket()` in network routing mode.

RF receiving

The `RFRXpacket()` function attempts to receive a packet and returns control to application after successful reception or after the timeout. Timeout during packet receiving terminates the reception except of the case if the *Wait packet end* option is enabled. The user has full control on timing as the timeout can be set (in ticks ~10 ms in STD RX mode or in cycles in LP and XLP RX modes, see below, *Low power modes*) prior to the `RFRXpacket()` (by `toutRF = ...`).

To manage power consumption, the following RX modes are available:

- STD: standard
- LP (Low Power): Receiving is combined with sleep mode. Incoming packets must be sent in LP TX mode (prolonging TX duration). Thus, power consumption during LP RX is lowered but consumption of the counterpart LP TX transmitter is increased.
- XLP (Extra Low Power): This mode is similar to LP but with RX staying in sleep for longer periods (see the diagram on next page). Incoming packets must be sent in XLP TX mode (even more prolonging TX duration). Thus, power consumption during XLP RX is even more lowered but consumption of the counterpart XLP TX transmitter is even more increased.

Routing in XLP is not supported in current OS version.

Resulting `RFRXpacket()` return value depends on the conditions (filtering, current network, RX mode, packet type, addressee etc).

After successful reception the respective values are valid: received data in `bufferRF`, data length (`DLEN`) and (in case of IQMESH) all other networking information.

To achieve higher noise immunity, incoming signal can be filtered for higher strength than the default level.

- In STD mode, `checkRF(x)` can be used before every `RFRXpacket` to ignore RF signal weaker than the specified level.
- In LP and XLP modes, `checkRF(x)` is intended just to specify the level for all subsequent `RFRXpacket` calls. It should be used only once whenever a change of filter level is needed. It should not be used repeatedly in RX loop.

Relative RF range is shortened due to this filtration. Refer to the datasheet of given TR (diagram *Relative range vs. checkRF*). For environment without a significant noise no additional filtration (`checkRF(0)`) is recommended.

See the IQRF OS Reference guide [1], `RFRXpacket()` and examples `E03-TR` (to see STD mode) and `E10-RFMODE-RX` (to see LP/XLP mode) [10].

Low power modes

RX and corresponding TX modes

STD RX

- Continuous receiving during `RFRXpacket`
- `toutRF` is in 10 ms ticks
- About 12 mA power consumption

STD TX

- 4 ms or 8 ms preamble (selectable by setRFmode)

LP RX

- In `RFRXpacket`, TR module is periodically switched between receiving (for a short period) and sleep.
  This defines a cycle = about 47 ms.
- `toutRF` is in cycles (see above)
- 250 µA average power consumption (TR-76D)

LP TX

- 50 ms preamble

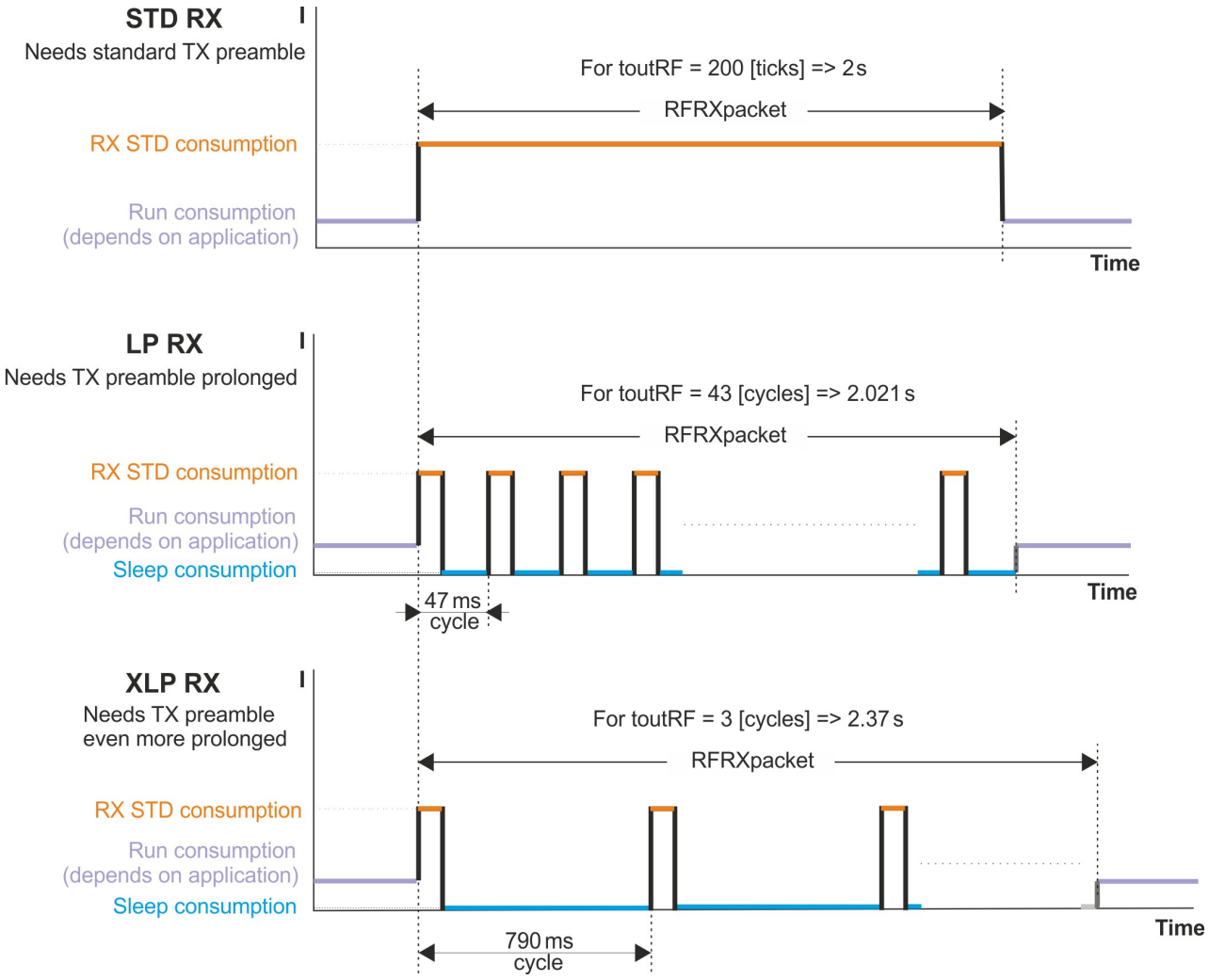
XLP RX

- TR module is periodically switched between receiving (for a short period) and sleep in `RFRXpacket`.
  This defines a cycle = about 790 ms.
- `toutRF` is in cycles (see above)
- 18 µA average power consumption (TR-76D)

XLP TX

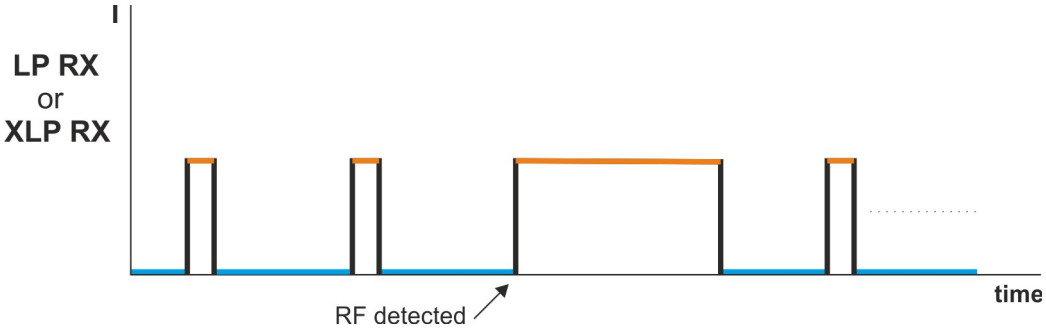
- 900 ms preamble

Waiting for incoming RF packets in STD, LP and XLP modes



Refer to the datasheet of given TR transceiver for actual power consumptions.

Response to incoming RF signal in LP RX or XLP RX



Thus, the following consequences are clear from above regarding RX modes:

- The only difference between LP and XLP is the sleep time duration (cycle period)
- TR module saves power only in the `RFRXpacket` function
- In LP and XLP, `checkRF(x)` is must not be used repeatedly in RX loop.
- Using of LP/XLP decreases RF range of 25%.
- Using LP/XLP in noise environment increrases power consumption (TR tries to receive a packet even when it is a noise).
- `setRFmode` offers a possibility to set stronger RF filter to increase immunity against a noise (but the range will be shorter)
- When the receiver is in LP/XLP mode the transmiter must prolong the preamble according to the RX mode. It is arranged automatically but the user must specify this in the `setRFmode` function.
- RX and TX modes can be combined in user application as needed. For example, a TR can receive in XLP RX and transmit in STD TX. It depends on the application and endurance of batteries.

The XLP mode in TR-7xD transceivers is supported for non-routing communication only. For routing communication it is more efficient to use LP mode combined with sleep as implemented in DPA framework (precise sleep).

Filtering

In case of chaining networks it can be selected whether packets should be received from both networks (including peer-to-peer packets) or from the current network only. If filtering is off current network is automatically switched to the network the packet was received from. See IQRF OS Reference guide [1], functions `setNetworkFilteringOn` and `setNetworkFilteringOff`.

Addressing

There are 2 types of addresses – see below (Routing, Addressing overview):

- Logical address: created by bonding.
- Virtual routing number, VRN: created by Discovery. For OS only. The user need not take care about VRNs at all.

Routing

Routing allows sending packets to addressees out of the sender's range using "hops" via devices which are in range each other.

IQRF routing is based on directional flooding of the network and TDMA (Time division multiple access) [12]. This IQRF OS supports up to 239 routing devices for a network. There are several routing algorithms specified in the `RTDEF` register according to network topology. A packet is routed via devices in specified order (routing vector) in defined time slots with specified period each. Retransmitting passes in reverse order. OS ensures that the packet is ignored by all devices except of the addressee and the routing devices.

For effective IQMESH the topology (placement of devices with respect to the range) should be designed in a redundant way - every device should have sufficient number of devices in range. Routing algorithm should be specified with respect to reliability and speed requirements. Due to time slots the efficiency should considerably depend on the order in the routing vector as well. The addressee does not route the packet except of a broadcast one.

Thus, routing allows higher range, lower RF output power, more ways to deliver packets, higher noise immunity, resistance against failures and dropouts (self-healing) and flexibility with respect to dynamic changes in range among individual devices (moving of persons, obstacles or devices themselves) which results in better throughput and reliability.

Routing can be enabled or disabled for individual nodes. Routed packets can be received whenever the `RFRXpacket()` is active in routing device but they can be retransmitted in respective time slots only.

Discovery

Nodes can be placed according to their addresses (with respect to fixed routing vector 1, 2, 3, …) or in a random order. But the random order requires Discovery when internal routing backbone is created and routing paths are found automatically.

Discovery assigns VRNs to Nodes and assorts them to zones (groups of Nodes which can be reached by the same number of hops from the Coordinator). See animation at www.iqrf.org/technology/iqmesh/iqmesh-animations/discovery.

Nodes intended to work as routers must be bonded from address 1. Function `discovery(MaxNodeAddress)` has the parameter specifying maximal logical address of the Node to be discovered. Therefore, the application should have separated address areas dedicated to routers (1 to `MaxNodeAddress`) and non-routers (`MaxNodeAddress+1` to 239 or less).

Example:
- Addresses 1 – 50 dedicated to routers
- Addresses 51 – 100 dedicated to non-routers

Routers must be bonded successively from address 1 (without vacations). If an address is omitted, the discovery process will uselessly take a long time due to attempts to discover the Node with missing address. Thus, for 20 bonded Nodes (with addresses 1 – 20) the `discovery(20)` should be called.

Discovery must also be called after adding or removing any routing device(s) by `setRoutingOn()` or `setRoutingOff()`.

Only discovered Nodes can work as routers (having VRN assigned).

Number of discovered nodes can be less than number of bonded nodes. In spite of this, the network can be functional. After Discovery, it is recommended to check connectivity to all Nodes (ping). If the communication is reliable, the lower number of discovered Nodes itself means lower number of routers but may need not worrying.

It would be usefull to run Discovery with lower RF power. Current RF power used by the Coordinator when Discovery is called is used also for all Nodes participated in Discovery process (possible restoration is up to the user). If Nodes are discovered even with lower RF power than the one used in standard operation, it is supposable that the communication will be robust and reliable.

To allow a Node to be discovered, it must have routing enabled (by function `setRoutingOn`) and the `answerSystemPacket` function must be running in its receiving loop.

The duration of the Discovery process depends on the topology, number of discovered Nodes and their accessibility. It may take tens of seconds up to tens of minutes.

Discovery data is stored in serial EEPROM. Refer to the Appendix 4 for the data mapping.

After changes influencing the range (changes in topology, addressing, device placement, obstacles, permanent failure of a router etc.) the Discovery should be reinvoked.

Routing is possible under all the following conditions:
- The routing device is bonded to respective network
- The packet was sent by the original sender with routing requirement (`ROUTEF = 1`)
- The `RFRXpacket()` function is active in the routing device when the packet to be routed is sent.
- The `ROUTEF` flag relates to outgoing packets and has no influence to routing incoming packets at all.

Due to power consumption it is recommended to supply routers from mains adapters. Battery operated routers should use the LP receive mode.

Every application has usually very different requirements. For example, a typical Smart House application can be realized with 4 hops and there is a need for fast response, while collecting data from power meters usually needs a network supporting much more hops but the latency is allowed. Thus, IQMESH specification supports various routing algorithms.

LP discovery

The discovery works also for Nodes in LP RX mode.

Coordinator:
On the Coordinator side, the handling is the same for both STD or LP discovery. LP discovery is selected if the LP TX mode is selected in the Coordinator when discovery is called.

Node:
Nodes can run either in STD or LP RX mode. The only difference between STD and LP discovery on the Node side is that Nodes in STD RX must have the `_ignoreForcedRoutinLP` bit cleared (to enable Forced LP routing, see below).

LP discovery can take slightly longer time.

Forced LP routing

Every routing device routes packets according the TX mode (STD/LP) of the routing device itself.

Every routing device routes packets in LP TX mode (independently on the TX mode of the device itself) if the packet has originally been sent in LP TX mode. Then the device returns to the TX mode having before routing such a packet.

To enable this forced LP routing, flag `_ignoreForcedRoutingLP` must be cleared. It is arranged by OS be default.

Forced LP routing allows to operate Nodes in STD RX as well as in LP RX modes in one network (e.g. to combine devices supplied from mains and from batteries).

If forced LP routing is enabled, the time slot duration (`RTTSLOT`) corresponding to LP must be used by all Nodes. See the explanation on next page.

IQMESH routing rules

- Every node routes a packet only once.
- Every node routes in the time slot corresponding to the address of the node, either logical (for SFM routing) or VRN (for DFM routing). See chapters Routing algorithms and IQMESH in practice.
- The Coordinator and the addresse does not route at all.
- Routers should not be moved (static routing backbone). After moving a router the discovery should be reinvoked.
- Not all nodes must be assigned to be routers.
- IQMESH supports communication between the Coordinator and a node only. Synchronous communication is recommended: requests initiated by the Coordinator and answers from nodes. Asynchronous packets are also allowed but they must be completely managed by the user. Possible collisions must be avoided in application software. It is not allowed to send routed asynchronous packets from not discovered Nodes at all.

All algorithms works with the following parameters:

- `RTDEF`: routing algorithm. See chapter *Routing algorithms*.
- `RTHOPS`: specifies number of hops per packet (0 – 239). 0 means direct delivery (single hop in all cases, without routing), e.g. 2 means typically 3 hops (3 packets sent: sender + 2 routers).
  The exact behavior regarding hops is as follows:
  - `RTHOPS` means 1 + (number of routing hops `- 1)`, in case of:
    - Unicast, without optimizing, if `RTHOPS` >= VRN of the addressee.
    - Unicast, when the addresse is a discovered Node in DOM routing algorithm (`optimizeHops (0xFF)`), see below.
    - Unicast, when the addresse is the first discovered Node (the lowest VRN) in given zone in DRM routing algorithm (`optimizeHops (0x00)`)

    In this cases the addressed Node does not transmit, `RTHOPS` is decremented just to 1 (not until 0) and the loop waiting for the end of routing waits the last time slot unnecessarily.
  - `RTHOPS` means 1 + number of routing hops, in case of:
    - Broadcast
    - Unicast, without optimizing, if `RTHOPS` < VRN of the addressee.
    - Unicast, when the addressee is a not discovered Node.
    - Unicast, when the addressee is not the first discovered Node (not the lowest VRN) in given zone in DRM routing algorithm (`optimizeHops (0x00)`).

  The user can set number of hops according to specific needs. But the maximal reasonable value is the number of routing nodes in the network.

  Flooding: If an RF packet is sent with `RTHOPS` = number of routers, it is called flooding. Every router resends the packet in dedicated time slot. Flooding is the most robust but the most time consuming communication. It is necessary to use flooding for communication with a moving node (not knowing routers in range). If all nodes in the network operate also as backgroung routers it is possible to set `RTHOPS` to the number of bonded nodes: `RTHOPS = eeReadByte(0).RTHOPS` is decremented in each hop (arranged automatically by OS).

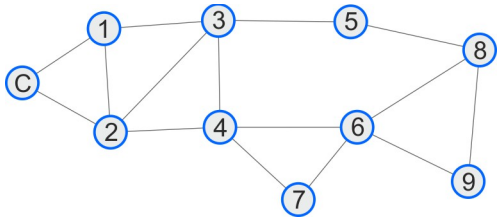
  `RTTSLOT`: time slot duration (in ticks). It should be longer than the transmit time for given packet. It depends on number of user data to be sent, `PIN`, `DLEN`, RF mode and RF speed. DPA sets the proper time slots automatically. Refer to *IQRF DPA Framework Technical guide*.

If number of hops = number of bonded nodes the routing is very robust (flooding). But even flooding is no assurance that the packet is successfully routed (overdue if unsuitable order of Nodes – see chapter IQMESH in practice).

Routed packet is delivered in `frame = time slot length x number of hops` (`RTTSLOT x RTHOPS`) and **must not be answered until the frame is elapsed** otherwise a collision with routing devices occurs. Time starts from the packet sent by the Coordinator (the first slot is dedicated to the Coordinator).

Routing algorithms

- **SFM (Static Full MESH)**
  SMF algorithm is intended for cases where the sender knows the network topology (the paths to the addressee). Up to 240 devices in a network is allowed. Routing vector is fixed (1, 2, …, 239), logical addresses are used for addressing (`RX` = logical address). Broadcast address is 0xFF. Bonding can be done before placement, addresses must be known and devices must be placed with respect to topology (adresses should increase with the distance from the Coordinator). There is no reason to perform Discovery in this case.

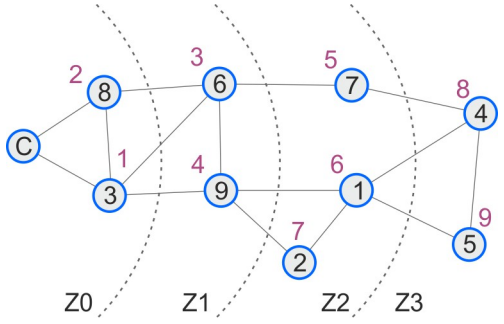


  If the logical address of a node is greater than number of hops specified in `RTHOPS` of given packet, it will not be routed by this node. Thus, a node can be excluded from the routing structure by assigning the logical address out of the address space dedicated to routers. In SFM, the `setRoutingOff()` function does not exclude the node from routing. Excluded nodes can be placed in random way but they must be in range with some router(s).

  Setting of number of hops (`RTHOPS`):
  - For a packet from the coordinator to a node: `RTHOPS` = number of bonded Nodes working as routers
  - For a packet from the node to the coordinator (answer):
    - For routing nodes: `RTHOPS` = logical address of the sender
    - For non-routing nodes: `RTHOPS` = the highest logical address of all routers in the network.

  Example:
  A network containing 100 nodes. According to given topology, just 20 routers are enough for sufficiently robust coverage. These routers are bonded with logical addresses 1 to 20 and placed in ascending order from the coordinator. The other nodes are bonded with addresses 41 to 120 and placed in random way. (Therefore, there are 20 free positions in address space of routers reserved for possible increasing of number of routers in the future.) For answer, every router has set `RTHOPS` = its' own logical address and all non-routing nodes `RTHOPS = 20`.

- **DFM (Discovered Full MESH)**
  Up to 240 devices in a network is allowed. Random placement is allowed and addresses need not be known for routing purpose but Discovery has to be performed after placement and bonding. Routing uses renumbered addresses (VRN, Virtual Routing Numbers) as a result of the Discovery process which creates a routing backbone with up to 240 devices devided to zones (based on minimal number of hops to individual Nodes). VRNs increase with the distance from the Coordinator (virtual routing backbone), are intended for OS only, the user need not take care about it. After a change in topology Discovery should be repeated. DFM is analogic to SFM but VRNs are used for routing instead of logical addresses. User addressing is completely the same (`RX = logical address`). Broadcast address is `0xFF`.



  1 – **9**: logical addresses, 1 – 9: VRNs, Z0 – Z3: zones. See the picture above and animation at www.iqrf.org/technology/iqmesh/iqmesh-animations/discovery.

  To disable routing for a node:
  - Call the `setRoutingOff()` function. Then no VRN will be assigned to this node during Discovery.
  - Do not call the `answerSystemPacket` function.

  Setting of number of hops (`RTHOPS`):
  - For a packet from Coordinator to Node: `RTHOPS` = bonded Nodes working as routers (flooding)
  - For a packet from Node to Coordinator (answer):
    - For routing nodes: `RTHOPS = 0xFF`.
    - For non-routing nodes: `RTHOPS = 0xFF`. Then OS sends the packet to the device (either the Coordinator or the Node) which received the request from.

- **DOM (Discovered Optimized MESH):** DOM is a special case of DFM. It is quite the same but number of hops is optimized (reduced) by the `optimizeHops()` function as follows: It sets the `RTHOPS` (number of hops) according to Discovery results to VRN of addressed Node to speed up the transmission at the cost of reduced redundancy.
- **DRM (Discovered Reduced MESH):** DRM is another special case of DFM. It is quite the same as DFM but number of hops is optimized (reduced) by the `optimizeHops()` function as follows: It sets the `RTHOPS` (number of hops) according to Discovery results to VRN of the first Node in the zone of the addressed Node. Compared to DOM, this brings further speeding up the transmission and lowering the redundancy.

Addressing overview

Routing algorithm	RTDEF	Addressing by	Routing by
SFM	0x01	Logical address	Logical address
DFM	0x02	Logical address	Virtual address

Fast Response Command

FRC (Fast Response Command) is a process allowing to quickly collect designated information and data from multiple Nodes in the network. It is much faster (even by orders of magnitude) than polling individual Nodes one by one.

Examples for times needed to collect responses from Nodes:

Nodes	FRC time	
	STD	LP
10	1.88	2.52
239	33.94	46.03

These times are valid for `_FRC_RESPONSE_TIME_xxx_MS` = 40 ms. For other `_FRC_RESPONSE_TIME_xxx_MS` values the times should be prolonged for corresponding delay. See the text below and IQRF OS Reference guide [1], `sendFRC` preconditions.

FRC is invoked by the Coordinator by calling the `sendFRC(cmd)` function which sends a request (possibly specified closer by the `cmd` parameter) via a broadcast packet (intended for all Nodes, either discovered or not discovered). All Nodes should respond to this request by the `responseFRC()` function which automatically delivers given application data from each Node to the Coordinator in very fast way.

Currently supported formats of application data to be collected:
- **2 bits** from up to 239 Nodes (with logical addresses 1-239)
- **1 byte** from up to 63 Nodes
  For not selective FRC: from nodes with logical addresses 1-63
  For selective FRC: from up to 63 nodes selected from 239 Nodes
- **2 bytes** from up to 31 Nodes
  For not selective FRC: from Nodes with logical addresses 1-31
  For selective FRC: from up to 31 Nodes selected from 239 Nodes
- **4 bytes** from up to 15 Nodes
  For not selective FRC: from Nodes with logical addresses 1-15
  For selective FRC: from up to 15 Nodes selected from 239 Nodes

Data collected from the Nodes is stored in the `bufferINFO` in the Coordinator.

For proper FRC operation the definite rules must be observed. See IQRF OS Reference guide, functions `sendFRC` and `responseFRC`.

FRC fits best for applications where the same type of information should be collected from each Node. E.g. for street lighting it is possible to switch all lamps ON by a single packet and then immediately collect status information from each lamp. So FRC is much faster than a successive polling of individual Nodes.

FRC timing

FRC allows to transfer 30 B from the `DataInSendFRC` array to the `DataOutBeforeResponseFRC` array of the Nodes. FRC can operate in either STD or LP TX mode.

`sendFRC()` blocking time is:

STD mode: `BONDED_NODES * 30 + (DISCOVERED_NODES + 2) * 110 + _FRC_RESPONSE_TIME_xxx_MS + 220 [ms]`
Thus, the the worst case for 239 Nodes is lower than from 33.94 s to 54.42 s (depending on the `_FRC_RESPONSE_TIME_xxx_MS` time needed to require response data by Nodes).

LP mode: `BONDED_NODES * 30 + (DISCOVERED_NODES + 2) * 160 + _FRC_RESPONSE_TIME_xxx_MS + 260 [ms]`
Thus, the the worst case for 239 Nodes is lower than from 46.03 s to 66.51 s (depending on the `_FRC_RESPONSE_TIME_xxx_MS` time needed to require response data by Nodes).

However, to estimate total FRC duration (e.g. in a higher system like a gateway or a cloud), besides of this blocking time, the user should take also possible additional delays for handling one's own FRC-relating overhead (possible buffer copying, SPI communication etc.) into account.

Selective FRC

Non-selective FRC requests FRC response from all Nodes (with addresses from 1 up to 239). Selective FRC requests FRC response from selected Nodes only. Nodes from 1 up to 239 can individually be selected by the bit array (in `bufferINFO`) in the Coordinator.

Selective FRC is useful for fast response from specified group of Nodes.

FRC configuration

FRC functionality is configured by the `configFRC` control register in the Coordinator:

`configFRC` Address `0x5B7`

7	6	5	4	3	2	1	0
	R/W	R/W	R/W			R/W	R/W
	FRCresponseTime				_4ByteFRCmode	_2ByteFRCmode	_selectiveFRCmode

`FRCresponseTime` specifies maximal time needed to require response data by Nodes (in ticks):

FRCresponseTime	Ticks
0	4
1	32
2	64
3	128
4	256
5	512
6	1024
7	2048

`_selectiveFRCmode`
- 0: Non-selective FRC
- 1: Selective FRC

`_2ByteFRCmode`
- 0: 2 b or 1 B response is requested, depending on the bit 7 in sendFRC parameter:
  - 0: 2 bits collected
  - 1: 1 B collected
- 1: 2 B response is requested.

`_4ByteFRCmode`
- 0: 4 B response is not requested
- 1: 4 B response is requested.

Refer to IQRF OS Reference guide, function `sendFRC`.

Bonding

In case of network communication, the network must be created first. It means that all Nodes to be included in the network must be assigned (paired) to given Coordinator. This procedure called bonding is a basic step of the network installation process. Actual implementation of bonding procedure depends on the user and should be designed with respect to specific application. Before creating a new IQMESH network (before the first bonding), the Coordinator must be initialized by the `clearAllBonds()` function.

Bonding is accomplished over the air by exchanging some system RF packets. To avoid bonding of an unintentional device and to hide sensitive data such as NID (Network ID) and Network password passed to the new device within the bonding, all bonding communication is encrypted by Access encryption, excluding the Smart connect bonding which is encrypted by IBK, see below. Thus, except of the Smart connect, all devices to be included in given network must have set the same Access password as the Coordinator and possibly the device providing prebonding. See chapter *Access encryption*.

Except of the Remote bonding, the Node to be bonded need not know (specified in TR configuration) the RF channel for communication in given network (the operating channel after bonding). Bonding utilizes fixed service channels and the bonded Node inherits the operation channel from the Coordinator within the bonding procedure (and write it into the TR configuration of the Node).

Bonding can be performed “on the desk” (before final placing of devices) or at real location (after final placing of devices).

Bonding results in storing the following information in EEPROM:

- On the Node side:
  - Logical address (assigned by the user) from 1 to 239. (Address 0 is reserved for the Coordinator, address 255 for broadcast packets and 254 for universal address for prebonded but not yet authorized Nodes).
  - NID (Network ID) assigned by the Coordinator based on the unique serial number (Module ID) of the Coordinator.
  - Network password assigned by the Coordinator.
  - A flag indicating bonding (informing the Node itself that it is bonded when asked by the `amIbonded` function)
- On the Coordinator side:
  - MID (Module ID) of bonded Node.
  - Corresponding flag in the bit array of bonded Nodes.
  - Incremented number of bonded Nodes.

A bonded Node is allowed to receive packets from its own network only and in case of disabled filtering by the `setNetworkFilteringOff()` function also non-networking packets.

There are three ways of bonding:

- Local — Bonded Node is in direct range with the Coordinator
- Remote — Bonded Node can be out of direct range with the Coordinator but in range to an already bonded Node.
- Smart connect — Bonded Node can be out of direct range with the Coordinator but in range to an already bonded Node.

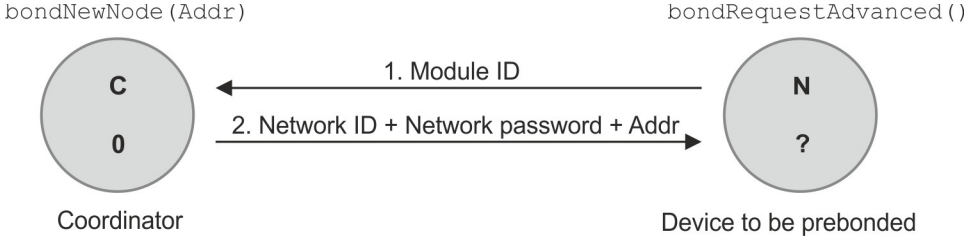
All devices taking part of the bonding process work in STD or LP RX modes (according to the RX mode used in given network), excluding Smart connect (see below) which always uses LP RX mode only.

Local bonding

Applicable for a Node to be bonded in direct range with the Coordinator.

Steps:

- Call the `clearAllBonds()` function in the Coordinator before the first bonding.
- Call the `bondNewNode(x)` function in the Coordinator with desired logical address as the parameter. This switches the Coordinator in receiving of system RF packets for cca 10 s period.
- While the `bondNewNode()` function is active in the Coordinator, call the `bondRequestAdvanced()` function in the Node. This function sends one system RF packet with request to bond and then waits for some time for the confirmation. If the request is confirmed the bonding is accomplished.



Remote bonding

Remote bonding allows to bond even Nodes out of direct range with the Coordinator but in range with at least one already bonded Node. This process consists of two parts: Prebonding and Authorization.

Prebonding

Already bonded Node(s) or the Coordinator itself (further on as the “device(s)” in the chapter *Remote bonding*) can provide so called prebonding to the Node(s) that is/are not part of the network yet.
Successful prebonding results in assigning a temporary (universal) logical address 254 to the new Node. The new Node also receives NID thus it is able to be part of the network although it has only a temporary address. It is possible to successively prebond more Nodes which then create a group of prebonded Nodes.

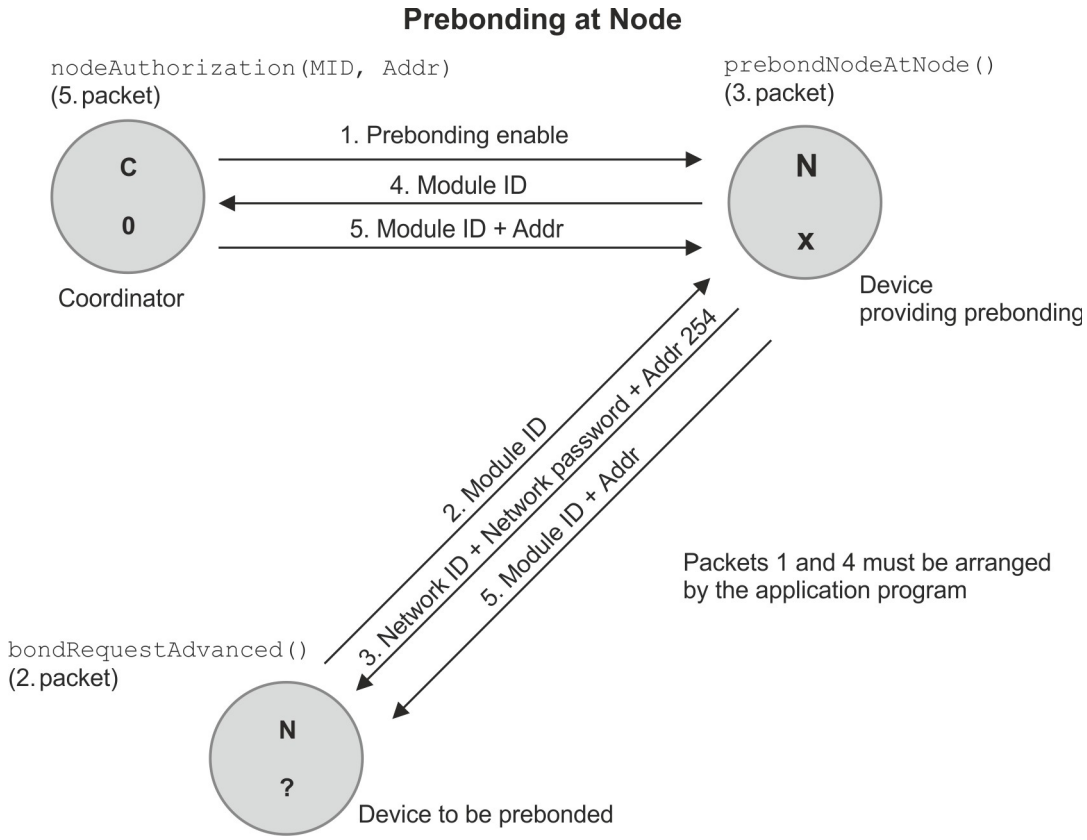
During prebonding, 4 B user values are exchanged between the prebonded Node and the device that provides prebonding:

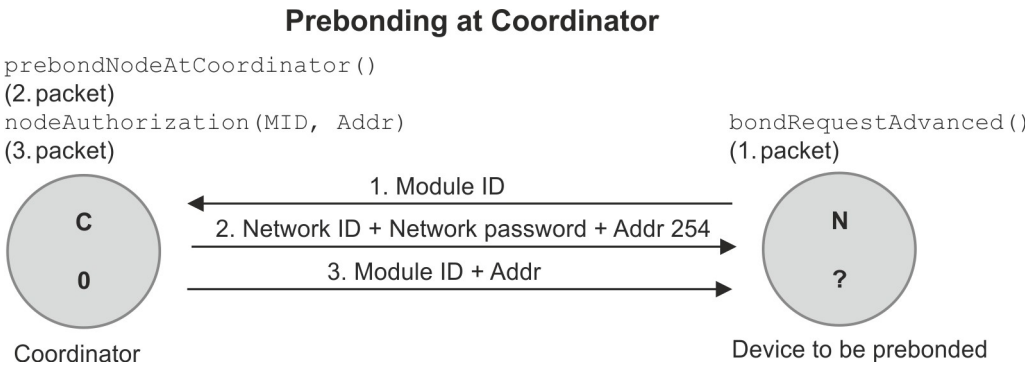
Prebonded Node		Device that provides prebonding
`nodeUserDataToSend`	→	`nodeUserDataReceived`
`hostUserDataReceived`	←	`hostUserDataToSend`

Until being authorized, the prebonded Node is allowed to receive and transmit system packets and respond to authorization FRC commands only. It is not allowed to participate in regular network communication.

Authorization

To complete the remote bonding, the Coordinator must assign a requested address to every prebonded Node. The MID of the prebonded Node must be known by the Coordinator to do it. It must be delivered to the Coordinator by the application program (sent via the network, collected by a service tool etc.). The MID is used during this process (called Authorization) as the pseudo-address of given Node in the group of all prebonded Nodes all having same temporary address 254.





Steps

- Devices selected to provide prebonding must have this feature active by the `prebondNodeAtNode` or `prebondNodeAtCoordinator` function called in RF RX loop. (This calling can be conditional, depending on the application).
- The Node to be prebonded must invoke the `bondRequestAdvanced()` function. This function sends one system RF packet with request to bond and then waits for some time for the confirmation by the device that provides prebonding. The request is sent in the TX mode (STD or LP) currently selected in the requesting Node (by the `setRFmode` function). This should correspond to the RX mode of given network. If the request is confirmed then the NID, Network password and the address 254 is assigned to the prebonded Node and its MID is stored in `bufferINFO` of the device that provides prebonding after the `prebondNode()` function call is finished. At this time the prebonding process is completed.
- The application program must find out whether a device providing prebonding has prebonded a new Node. If so, the MID of this new Node must be delivered from the Node that provides prebonding to the Coordinator. This is no use in case of prebonding by the Coordinator.
- Then the Coordinator should call the `nodeAuthorization()` function. Parameters of this function are the MID of given Node and requested logical address. This function sends a system packet to the network.
  - In case of prebonding at the Coordinator, the authorizing packet is sent directly to prebonded Node without routing which significantly speeds up the authorization.
  - Otherwise the authorizing packet should be forwarded via the prebonding Node.

  To enable authorization of the prebonded Node, it must have the `answerSystemPacket()` function running in the RF RX loop.
- By receiving the system (authorizing) packet the authorization is accomplished automatically and then the Node is definitely bonded having the requested logical address. However, IQRF OS does not check the result. Thus, successful authorization should be verified by the application program (ping, Discovery etc.).

Refer to the IQRF OS Reference guide for details describing individual bonding functions.

The user can check results and make arbitrary changes in bonding at any time. There is a set of OS functions dedicated to bonding and related operations (access results, unbonding, rebonding etc). But once the Node is bonded and respective records are written to EEPROMs on both sides, the Coordinator as well as the Node start keeping their own bonding information independently to each other and no subsequent changes in bonding are carried over to opposite side via RF automatically arranged by OS.
In short, only `bondRequestAvanced()`, `bondNewNode()`, `prebondNode()` and `nodeAuthorization()` exchange RF system packets between Coordinator and Node. All subsequent changes in bonding by either Coordinator or Node are written to EEPROM just on one side. Functions `removeBondedNode()`, `rebondNode()` and `clearAllBonds()` operate with the Coordinator bit array only and `removeBond()` operates with the Node flag only. If synchronization between the Coordinator and the Node after changes is needed it must be done by the application program. Static systems that suit IQRF best have moderate requirements for changes in bonding.

Bonding data is stored in MCU internal EEPROM and in serial EEPROM. Refer to the Appendix 4 for the data mapping.

Smart connect

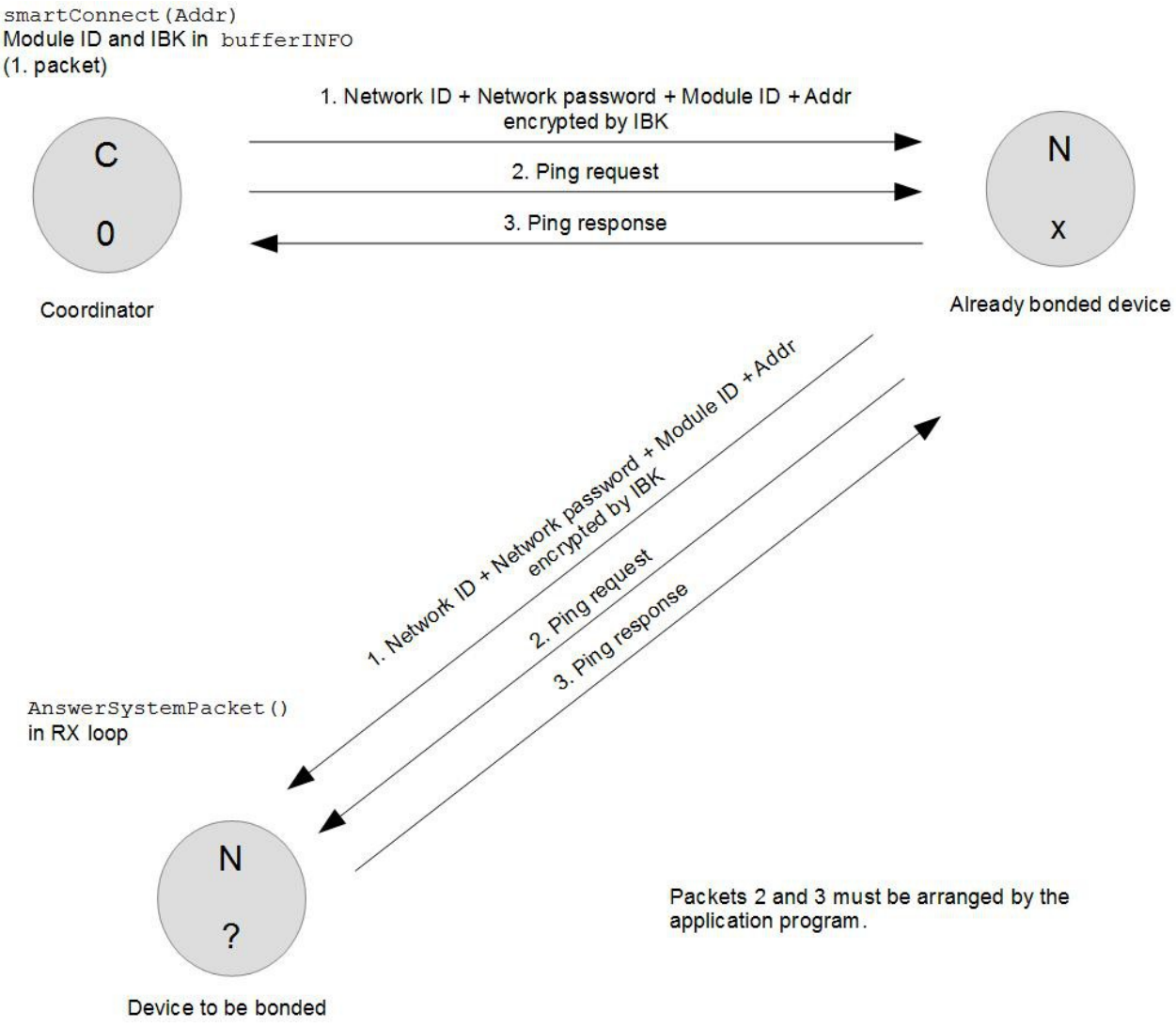
Smart connect allows to bond even Nodes out of direct range with the Coordinator but in range with at least one already bonded Node. Smart connect is a bonding method requiring **no Node configuration** and **no action** (e.g. a pushbutton press) **on the Node side** for bonding. It is managed by the Coordinator.

This bonding is based on the **IBK** (Individual Bonding Key), a unique and fixed code assigned and stored to every TR from the factory. To add a Node into the network, just the Coordinator's knowledge of the Node's IBK and **MID** (Module ID) will suffice. Both these values must be **imported into the Coordinator** in advance. Nothing is necessary to specify on the Node side for bonding (Access password and RF channel need not match the Coordinator values).

During the Smart connect, the Coordinator sends a system broadcast packet using the 3 service RF channels (dedicated to Smart connect and some other system communication, see *Appendix 2 – Channel maps* for RF frequencies). It is encrypted by the IBK of the Node being bonded. This packet is routed through the network by all routing Nodes but is only understandable by the Node with given IBK (which must of course be unbonded at that time). To be bonded, this Node must only call the OS function `answerSystemPacket()` in LP RX repeated loop.

During Smart connect, 4 B user values are exchanged between the Smart-connected Node and the Coordinator:

Smart-connected Node		Coordinator
`hostUserDataReceived`	←	`hostUserDataToSend`
`nodeUserDataToSend`	→	`nodeUserDataReceived`



The main precondition for Smart connect is the fact that the Coordinator knows IBK and MID of all Nodes to be bonded. Both of these values can be read out by the `moduleInfo()` function or in IQRF IDE (Ctrl+M). Transferring this data is up to the user application.

Steps

- Store the IBK and MID of the Node to be bonded into the `bufferINFO` of the Coordinator. See the IQRF OS Reference guide, function `moduleInfo()`.
- Call the `smartConnect(x)` function in the Coordinator with desired logical address as the parameter. This is a blocking function (not finished until the Smart connect procedure is completed).
- The bonded Node must call function `answerSystemPacket()` in LP RX repeated loop at that time.
- By receiving the system (Smart Connect) packet the process is accomplished automatically and then the Node is definitely bonded having the requested logical address. However, IQRF OS does not check the result. Thus, successful Smart Connect should be verified by the application program (ping, Discovery etc.).

Tip

IBK and MID can be imported into the Coordinator in the IQRF Smart connect format. It is open and can easily be generated and imported in IQRF IDE. The IQRF Code specification and an encoder/decoder source code are freely available in a single bundle.

Example

Each device (Node) has a label with the QR code containing the *IQRF Smart connect code*. Within the installation, this code is read by the mobile application and passed into the gateway (the Coordinator).

Other bonding rules

- To avoid possible collisions, bonding should be requested (by the `bondRequestAdvanced()` function) just by one Node at the same time and the same area with direct range to bonding counterpart.
- However, this request can be received by more supporting Nodes (with activated prebonding) or even by the Coordinator (with activated function `bondNewNode`() at this moment). To avoid collisions due to multiple confirmations, the `bondingMask` register should be set in the appropriate way. Functions `bondNewNode()` and `prebondNode()` provide (pre)bonding only if the following internal condition is valid:
  `if (((bondingCounter ^ address) & bondingMask) == 0).`
  `bondingCounter` Request for bonding contains the counter cleared after reset and incremented before every `bondRequestAdvanced()` call.
  `address` Address of the Node (or the Coordinator) evaluating the condition above.
  Thus, proper selection of the mask ensures that every bonding request is processed by one device (either a Node or the Coordinator) only. The mask should be set with respect to logical addresses and number of devices in range being asked for the confirmation.

  Recommended values of the `bondingMask`:

Mask	Usage	Max. number of `bondRequestAdvanced()` calls for sure prebonding / bonding (valid in noise-free environment)
`0000 0000`	Default, no masking effect. Only one device with enabled prebonding is in range or local bonding by Coordinator is performed and no other device in range has enabled prebonding.	1
`0000 0001`	The network contains 1 – 2 Nodes (addresses assigned continuously) and more than one Node in range has enabled prebonding.	2
`0000 0011`	The network contains 3 – 4 Nodes (addresses assigned continuously) and more than one Node in range has enabled prebonding.	4
`0000 0111`	–"– 5 – 8 –"–	8
`0000 1111`	–"– 9 – 16 –"–	16
`0001 1111`	–"– 17 – 32 –"–	32
`0011 1111`	–"– 33 – 64 –"–	64
`0111 1111`	–"– 65 – 128 –"–	128
`1111 1111`	The network contains 129 – 239 Nodes (addresses assigned continuously) and more than one Node in range has enabled prebonding. Or the network contains any Nodes with addresses not assigned continuously.	239

- Bonding is performed on 3 fixed service channels and with currently selected RF output power. Thus, it is possible to lower RF power just for bonding process to enable bonding in close distance only (e.g. for security during local bonding). See *Appendix2 – Channel maps* for frequencies of the service channels.
- Bonding functions use STD RX and TX modes according to the RX mode of given network.
- Bonding process uses handshaked communication. Under the influence of a noise etc. it is possible that a confirmation packet is not delivered correctly which causes that the bonding is considered as a success by one side while as a failure by the opposite side. Such a situation must be handled by the application program. E.g. the Coordinator can always test bonding and authorization of a new Node by subsequent check packet (ping).
- Function `isBondedNode()` provides the Coordinator with the information whether the Node with given address is bonded.
- Function `amIBonded()` provides the Node itself with the information whether it is bonded. This can be used in the application e.g. to handle the situation when a Node is not bonded.
- Prebonding supportive Node need not be in direct range with the Coordinator but additional routing Nodes can take a part in delivery.
- Logical address of the Node (1 – 239) can be changed to the universal address 254 by the `removeBondAddress` function. NID and Network password stay valid, that is why the Node is still accessible (but for authorization only) in the network. It is suitable e.g. if logical address of the Node should be changed. A new address can then be assigned by the `nodeAuthorization` function.

Unbonding

A bonded Node can be excluded from the network (unbonded) later on.

- During **bonding**, prebonding, authorization and Smart connect, given **Node communicates with the Coordinator**. This results in bonding information automatically updated **on both sides**.
- On the contrary, **unbonding** is always accomplished **locally** (separately on the Coordinator and the Node sides). **No system communication passes between the Coordinator and the Node being unbonded.** Thus, unbonding can be performed one-sidedly (on the Node side or on the Coordinator side) or two-sidedly (on both sides in synergy). If unbonding on both sides is needed, the synergy is neither done by OS but it is up to the user application.

To unbond the Node, the following OS functions are intended:

- On the **Node side**: `removeBond()`
- On the **Coordinator side**:
  - Single Node: `removeBondedNode()`
  - All Nodes in the network: `clearAllBonds()`. It completely clears the list of bonded Nodes which actually voids the network from the Coordinator side.

Caution
A Node which has ever been bonded to a network and should be bonded to another one, must definitely be unbonded first.

It is possible to unbond a Node in several ways:

- **Within the application** code inside the Node and/or the Coordinator (arranged in Custom DPA handler)
- If the Node is **accessible in the existing network**
  - **By DPA commands**. Corresponding ones are: *Remove bond*, *Remove bonded Node* and *Clear all bonds*.
  - In **IQRF IDE** connected to the Coordinator (*IQMESH Network Manager* / *Control* / *IQMESH* tab). See the IDE Help.
- If the Node is not accessible in the existing network but is **accessible for physical manipulation**: DPA provides a universal method. See the *IQRF DPA Framework technical guide*, chapter *Device startup*.
- If the Node is in **direct RF range** and it is possible to **reset** it: Using the **CATS** service tool in IQRF IDE (menu *Tools / CATS Service Tools / DPA Service Mode* tab).
  See the IQRF IDE Help and video tutorial: https://www.youtube.com/watch?time_continue=1&v=yq_9RzFrDFU.

Autonetwork

IQRF network can even be created fully automatically. This procedure (called Autonetwork) is available as a ready-to-use Custom DPA handler within the DPA framework. All Nodes to be bonded must have specified the same Access password as the Coordinator. Refer to the *IQRF DPA Framework technical guide*, chapter *Autonetwork*.

Another possibility is controlling the Autonetwork from a superodinate system. Respective routines are available within the IQRF SDK (Software Development Kit).

Transceiver replacement

This chapter is intended for networking applications only.

It is possible to replace a transceiver (either the Coordinator or a Node) in the network (e.g. when the transceiver failed) without the necessity to perform a new bonding and Discovery. Networking data can be restored from the backup into any proper piece of transceiver.

To enable restoring at any time in future, it is necessary to have a file with backup data and know the Access password which had been selected at the original transceiver when its backup was created.

Backup

The backup should contain the last data regarding the topology of the network. Thus, it should be created after every change in the network (first of all after adding or removing a Node) or after Discovery.

The backup can be created:

- In IQRF IDE:
  - Menu *Tools → IQMESH Network Manager → Control → Backup*, button *Backup*
  - Menu *Tools → CATS Service Tools → DPA Service Mode*, button *Backup*
- In DPA protocol:
  - The *Coordinator / Node* peripheral, command *Backup*

Backup data is encrypted by Access encryption. Refer to chapter *Access encryption*.

Restore

To restore the data from a backup file, the Access password at the new transceiver must be set the same as the Access password at the transceiver that was originally backed up.

Additionally, the new device must have uploaded correct files for given network (HWP plug-in, TR configuration with correct Access password and possible Custom DPA Handler) before the restoring.

The restoring can be done:

- In IQRF IDE:
  - Menu *Tools → IQMESH Network Manager → Control → Backup*, button *Restore* (for the Coordinator only)
  - Menu *Tools → CATS Service Tools → DPA Service Mode*, button *Restore*
- In DPA protocol:
  - The *Coordinator / Node* peripheral, command *Restore*

The following conditions must be met to make the transceiver backup fully functional:

- The original as well as the new transceiver have the same Access password.
- No network traffic comes from/to restored Coordinator during the restore process.
- The Coordinator device must be reset after the restoring is completed.
- It is not strictly necessary to run Discovery before the network is used after restoring. But it is recommended to do so because of possible piece-to-piece differences in RF range between the new and the original transceiver.

Encryption

Every wireless system is exposed to potential over-the-air attacks. IQRF OS v4.00 brings ultimate security based on industrial standards ensuring authorized access to OTA-flowing data. Special attention is primarily given to security of networks. IQRF utilizes AES-128 encryption, an industrial standard for wireless communication. Besides hiding sensitive data, data encryption increases consistency protection and prevents packets forging.

Three different protections based on AES-128 are applied:

- Networking encryption: All networking communication is encrypted.
- Access encryption: Another independent encryption is always applied while bonding and for possible subsequent network maintenance.
- User encryption: Moreover, payload data (for networking as well as non-networking packets) can optionally be encrypted by user-specific key to hide its information content.

Industry standard AES-128 is used in all the cases. It is the most common standard symmetric block cipher, operating with 128 b data blocks and 128 b long keys. Thus, the length of the plaintext to be encrypted (and ciphertext to be decrypted) must be a multiple of 16 B. The same key must be used for both encrypting and decrypting.

Compromising of keys is very frequent source of security problems. Therefore IQRF OS minimizes and protects also manipulation with keys - network and access keys are not known during physical manipulation as they are generated from respective passwords. Network password is generated randomly with high entropy and delivered encrypted to devices that are joining the network during bonding. This approach offers the following significant advantages:

- The user should take care about the passwords only but never handle with keys. The management and distribution of keys is completely handled by OS.
- The keys depend not on the passwords only but are modified by embedded hash functions.
- These two separated layers (with no simple direct relationship between passwords and keys) additionally increase the security.
- The keys are generated dynamically, varying in time, which is significantly more immune against attacks.
- The relationship between passwords and keys are different in different networks.
- Breaking the keys in one network has no impact on other networks at all.

Both Bonding and Networking encryption/decryption proceed within the packet consistency check routines. Unauthorized packets detected in the network are scratched similarly as packets corrupted e.g. by an RF interference.

Access encryption

Access encryption is used to secure all sensitive OTA network operations. First of all, it is used for bonding which is one of the most important procedures from the security point of view, deciding whether a device is included into given network or not. Thus, only authorized devices and users with valid Access password are allowed to join the network or to maintain network devices. CATS services, network backup and restore etc. also use Access encryption. See IQRF IDE Help.

Smart connect is also encrypted by Access encryption but IBK is used instead of Access password in this case.

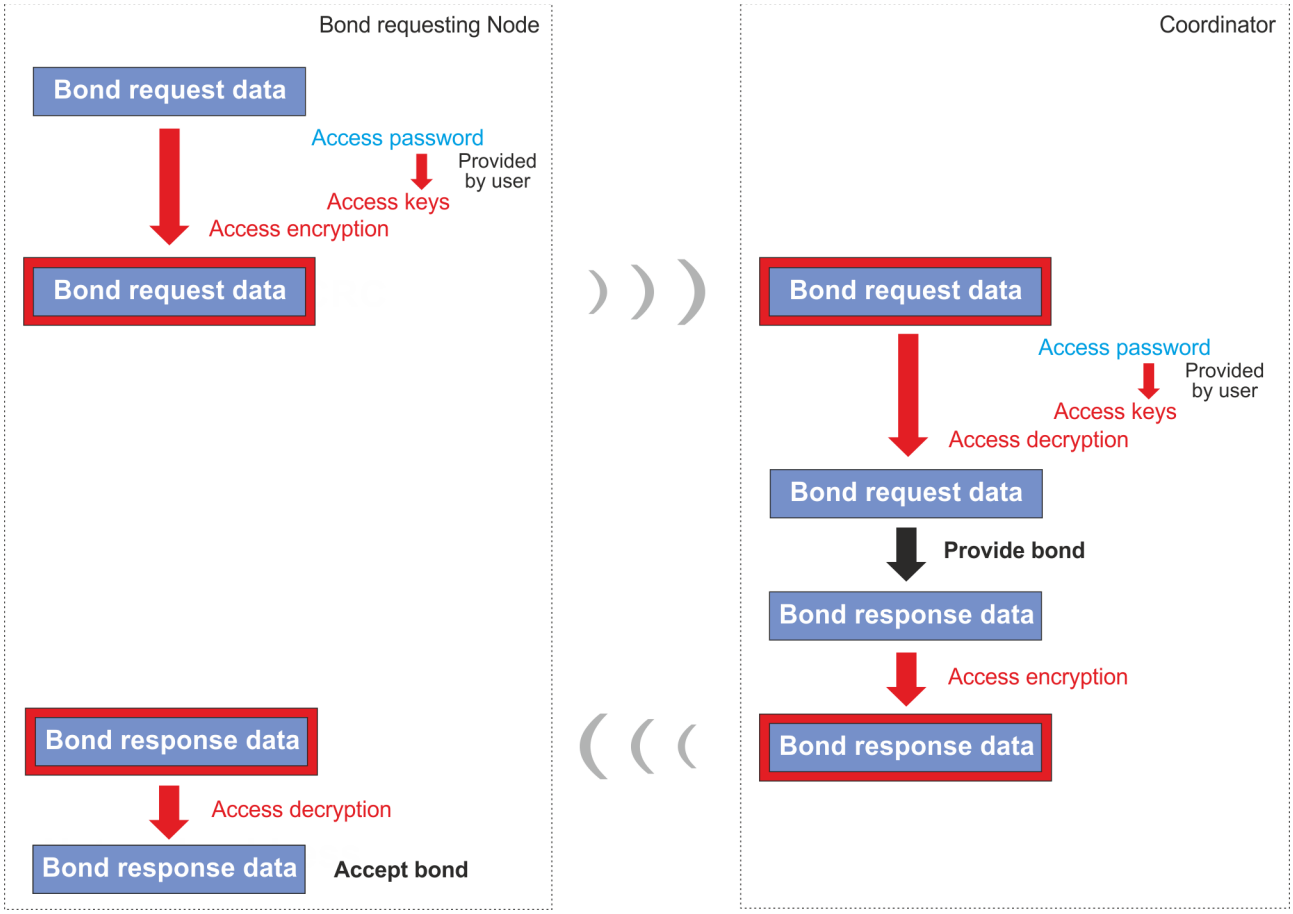
All sensitive data exchanged during bonding, such as networkPasswords, NID and Node address are transferred encrypted. Therefore, all subsequent network communication (which uses these data) described in next chapter is well secured.

AES-128 with 16 B long key and standard CBC mode of operation is used for Access encryption.

Generation of the Access key is based on the 16 B long Access password specified by the user. It must be the same for all TRs in given network. Default value set at all TRs from the factory is: `0000000000000000`. It is highly recommended to replace this default by a user-specific value, either in application SW by the `setBondingPassword()` function or in IQRF IDE *TR Configuration*.

See diagram *Bonding* in chapter *Encryption explanation examples* below.

Protection during bonding (simplified):



Networking encryption

Protection of standard network communication is the most important as it continues during the whole system life. All communication in IQRF networks via networking packets is encrypted. Therefore, only systems and users having valid Network password may join the network communication and process data. To avoid wrong implementations, all network communication has been forced to be encrypted automatically by OS and works quite transparently from the user's point of view.

AES-128 with 16 B long keys and additional proprietary CDC (Cipher Data chaining) algorithm is used for Networking encryption. Unlike Bonding encryption, the Networking password is not specified by the user. Every TR transceiver is equipped with a 192 b long unique random password individually generated at the factory. The 128 b long keys in given network are derived from the password of the Coordinator. The password is passed to Nodes securely, while bonding, encrypted by Bonding encryption. Therefore the user takes absolutely no care not only about the Networking encryption itself, but also about the management and distribution of network passwords.

User encryption

Besides protection of network communication, there is an optional way to increase privacy by adding another encryption shield to hide user's payload data (either networking or non-networking) or to enhance network protection. User encryption is fully under user's control. It utilizes User key specified by the user. Then only ciphertexts are transferred over the air which makes the information content not readable for systems or persons not knowing the User key.

See diagrams (*Non-)networking with User encryption* in chapter *Encryption explanation examples* below.

User encryption and/or decryption can optionally be performed outside TR transceivers which allows to hide user's data not only when OTA transferred but even from IQRF platform itself (partly or completely). See diagrams *Networking decrypted outside IQRF* and *Networking with double User encryption* in chapter *Encryption explanation examples* below.

Setting the User key

All TR transceivers participating in User encryption should have the User keys specified beforehand. The same key must be used for encryption as well as for decryption.

To set a User key, place desired 16 B value into `bufferINFO` and then call OS function `setUserKey()`.

Encryption

- At RF transceiver:
  To encrypt the data to be sent, call the `encryptBufferRF(blocks)` function. It is possible to encrypt the whole packet or a part only (specified as number of 16 B blocks to be encrypted). This function replaces the plaintext (16, 32, 48 or 64 B) in the `bufferRF` by the ciphertext encrypted by current User key. The rest of the `bufferRF` remains unchanged.
- Outside IQRF:
  The data to be send can also be encrypted outside TR transceiver and directly put into the `bufferRF`. If so, the encryption is up to the user, according to the following encryption parameters:
  AES-128 b, 16 B long key, CBC mode, IV = `0000000000000000`.

  Exact algorithm is specified in Advanced Encryption Standard specified at
  http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

  Recommended AES Calculator for testing of User encryption and decryption algorithms are
  http://extranet.cryptomathic.com/aescalc/index?key=00000000000000000000000000000000&iv=00000000000000000000000000000000&input=01234567890123456789012345678901&mode=ecb&action=Encrypt&output=66E94BD4EF8A2C3B884CFA59CA342B2E
  or a simpler one at http://testprotect.com/appendix/AEScalc.

.

Sending encrypted data

Encrypted data is ready to be sent by `RFTXpacket()`. See examples in IQRF OS Reference guide [1], function `encryptBufferRF()`. It is up to the user application to deliver the information whether the data is encrypted or not and possibly the number of encrypted blocks.

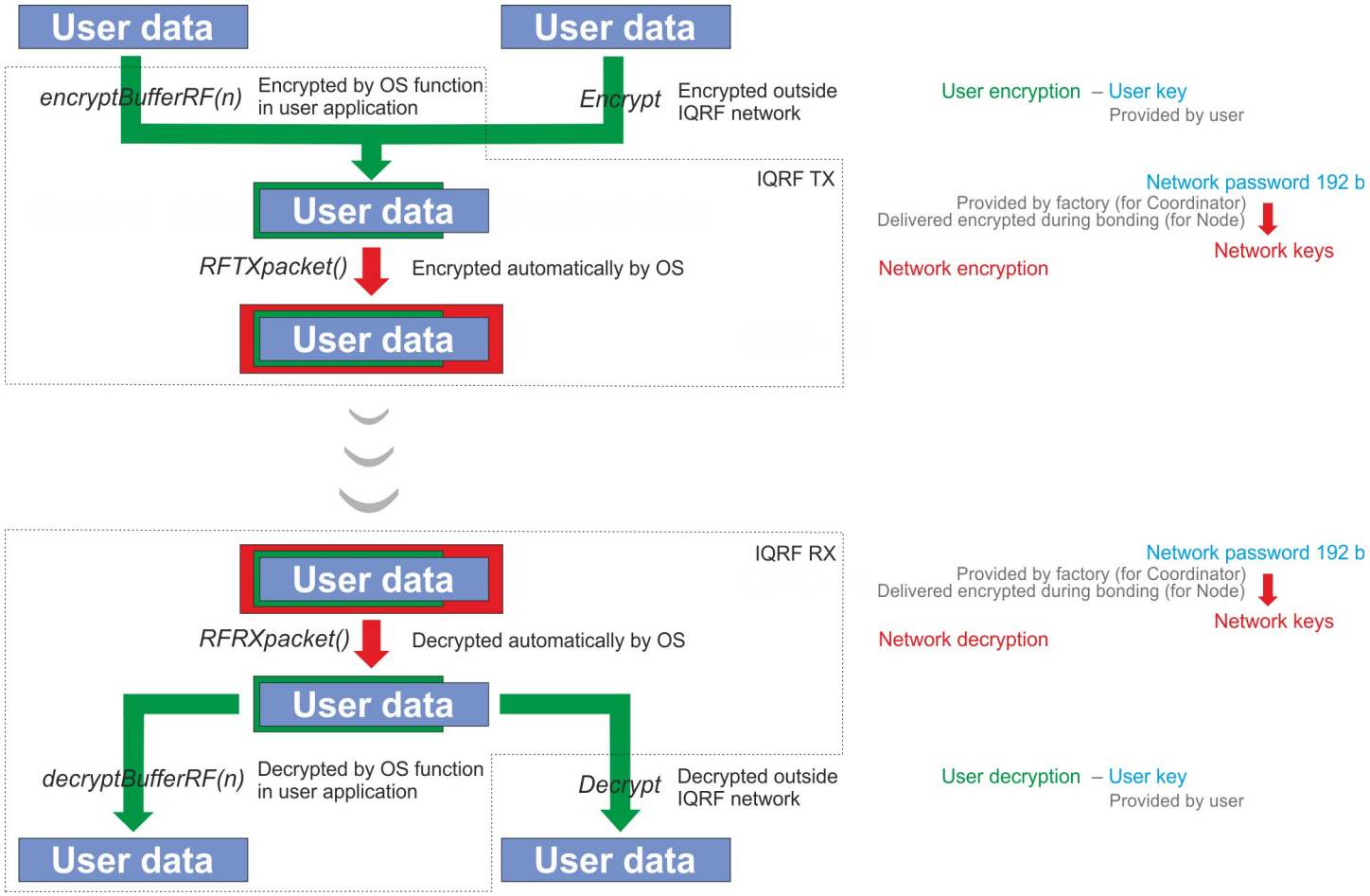
Receiving encrypted data

Unlike the Bonding and Networking encryption, OS provides no checking regarding the keys for User encryption. The packet is allowed to be successfully received even when the User keys do not match. The receiving device shoud know whether the packet (and possibly how many blocks) has been User encrypted. Managing of such information is up to the user application.

Decryption

- At RF transceiver:
  For decrypting received data at TR transceiver, the OS function `decryptBufferRF(blocks)` is intended. This function replaces the ciphertext (16, 32, 48 or 64 B) in the `bufferRF` by the plaintext decrypted by current User key. The rest of the `bufferRF` remains unchanged.
- Outside IQRF:
  However, it may not be required to process data within IQRF network. If so, the data can be left encrypted in TR transceiver and perform the decryption afterward, outside the IQRF, e.g. in a superordinate system or a cloud server. Then the decryption is up to the user, according to the same algorithm as for encryption, see above.

Protection during networking communication:



IQMESH in practice

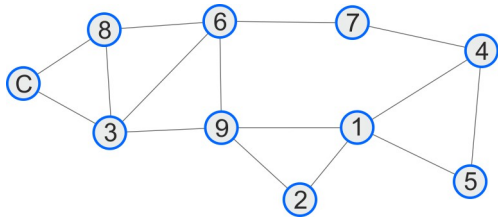
Routing explanation examples

Bonding

Let us have 9 Nodes bonded to a Coordinator. Every Node has a label with its logical address (1 to 9, assigned during bonding). These addresses are used for actual addressing.

Installation

Place the nodes to desired positions. Nodes in range each other are marked with interconnection lines on the figure. These links exist but are not known at the moment.



Sending a packet to Node 5

- C sends a packet. It is received by nodes N3 and N8.
- N3 waits 2 slots and then (in the third one) resends the packet.
- N8 waits 7 slots and then (in the 8th one) resends the packet.
- In slots 1 and 2 no packets are resent because nodes N1 and N2 have not received the packet.
- The packet resent by N3 in slot 3 is received by N8, N6 and N9. (C ignores the packet.)
- N8 receives the packet for the second time and is still waiting for its time slot.
- N6 receives the packet for the first time and is waiting for its time slot.
- N9 receives the packet for the first time and is waiting for its time slot.
- etc.
- In slot 9 N9 resent the packets to N1 and N2 but no one of them can route because their slots are already expired. Thus, destination N5 can not be reached from N1.
- Similarly, neither N4 can route the packet to N5 since it received this in slot 7 (from N7) after expiration (slot 4).

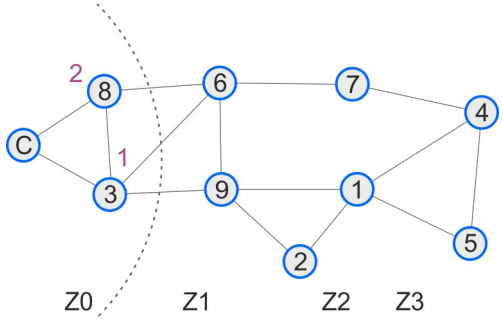
It is evident that N5 can never be accessed in such configuration and this MESH is not well designed. This MESH would work well if nodes are placed according to their logical addresses (ascending from the coordinator). This can be used in special cases only (e.g. street light lamps in a straight line without branches, with the Coordinator on one edge, **SFM**). In such cases the SFM routing algorithm (without Discovery) can be used.

In example above it is necessary to use the **DFM** routing. To utilize DFM, an additional step (Discovery) must be performed. From the source code point of view it means that to call the `discovery(MaxNodeAddress)` function and wait for the result returning number of discovered nodes. Duration takes some time (it can vary from tens of seconds to tens of minutes) depending on number of nodes, topology etc.

Discovery

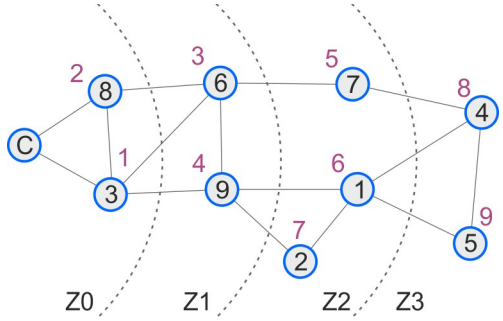
After the `discovery(MaxNodeAddress)` is called the following system communication on background is invoked:

- C sends a request packet: "Whoever is in range, answer me".
- N3 and N8 answers in this example. They are nodes in direct range from the Coordinator, creating the Zone 0 (Z0).
- C sends a packet to N3 (lower logical address has always priority): "You have assigned virtual routing number VRN = 1"
- C sends a packet to N8: "You have assigned VRN = 2"



- Now C passes control to N3 (lower logical address has also priority): "Discover your neighbors which have not been discovered yet, assign VRNs to them and give results back to me. The first free VRN is 3."
- N3 discovers new neighbors N6 (assigning VRN = 3 to it) and N9 (assigning VRN = 4).
- C passes control to N8 but it discovers no new neighbors.
- Now Zone 1 (N6 and N9) is discovered (nodes accessible from/to Coordinator via two hops.).
- C passes control to N6 which discovers N7 assigning VRN = 5 to it.
- etc.

Resulting renumbered network is arranged in ascending order according VRNs (DFM routing).



Now routing according the rule mentioned above will work. But the user does not know VRN addresses. It is not needed. To address nodes, the user uses logical addresses assigned by bonding.

Resume:

Routing algorithm	Addressing by	Routing by
SFM	logical address	logical address
DFM	logical address	VRN

The `discovery` function has the input parameter `MaxNodeAddress`. It is the maximum address of the node to be participating in the discovery process.. Return value is the number of factually discovered Nodes.

- If all nodes operate also as background routers, the most commonly `MaxNodeAddress` value is number of bonded nodes: `discovery(eeReadByte(0x00))`. (The Coordinator stores the number of bonded nodes in EEPROM at address 0.) This solves even the worst case when there is just one node in every zone – the Chain topology which is the worst MESH case (the less robust one) without redundant paths at all. But such an arrangement is not recommended.
- If some nodes do not route number of routers is used as the z parameter.

Example: Communication with N1. Topology see the picture above, DFM.

Packet from the Coordinator to the node (request):

```
...
setCoordinatorMode();      // To select the Coordinator and networking communication
RX = 1;
DLEN = 10;                 // Data are prepared in bufferRF
PIN = 0;                   // Preclearing
// _NTWF = 1; // Not necessary. This flag is set automatically by RFTXpacket().
//  It is natural that networking is concerned in case of Coordinator.
_ROUTEF = 1;               // To route outgoing packet (the PIN.5 flag)
RTDEF = 2;                 // DFM
RTHOPS = eeReadByte(0x00); // Number of hops = number of bonded nodes
                           // (only if all nodes operate also as background routers)
RTTSLOT = 2;               // 20 ms is enough for DLEN=10. See the formula at page 40.
RFTXpacket();
...
```

This code induces so called network flooding. It means that every discovered node (with a VRN assigned) routes the packet in its time slot. So routing vector 1, 2, 3, 4, 5, 6, 7, 8, 9 is used. Flooding is used especially in networks with moving nodes. They should not be in a routing structure. (It is not known which routers are in range. Moreover, it is changing in time.)

Besides of nodes routing in background, the network can also include nodes which do not route at all. For such nodes the following conditions should be kept:

- The `setRoutingOff()` function must be called once during initialization (default OS value is Routing On).
- The answerSystemPacket() function should not be called
- `setRoutingOn()` and `setRoutingOff()` are intended for DFM routing algorithms only.
- There is no reason to call the `wasRouted()` function.

N1 receives the packet four times in total:

- In 4. slot from N9
- In 7. slot from N2
- In 8. slot from N4
- In 9. slot from N5

Note: Addressed Node does not route.
Thus, the connection with N1 is quite robust. E.g. after failure in 4. slot the packet is still received in 8. and 9. slot.

In case of flooding the communication takes time according the formula:

Total time = time slot duration x number of hops (`RTTSLOT x RTHOPS`).

If the answer is required the time is twice longer.
IQMESH offers speeding up using optimizing number of hops – DOM (Discovered Optimized MESH). The optimizeHops() function called before sending a packet sets the number of hops (register `RTHOPS`) to specified value. Parameter 0xFF means to copy VRN of addressed node to `RTHOPS`.

```
...
optimizeHops(0xFF);     // DOM
RFTXpacket();
...
```

After this optimizing number of hops is reduced to 6 (including a packet from the Coordinator). Only nodes with VRNs 1 to 5 route packets. Routing vector is 1, 2, 3, 4, 5. N1 receives the packet only once (in 4. slot from N9).
Optimizing leads to faster communication (less time slots) but the robustness is reduced due to less redundant paths. Routing method should be specified according to particular needs of an application. E.g. a packet can be sent with optimization at first and if no answer is returned the packet is sent once more not optimized.

Packet from the Node to the Coordinator (answer):

Registers `RTDEF`, `RTHOPS`, `RTTSLOT`, … are directly included in RF packet. That is why the routing algorithm, current time slot, number of hops and time left to end of routing are known for every node which has received the packet. OS automatically decrements `RTHOPS` in every hop. This is used for answers. Addressed node can receive the packet (RFRXpacket returns 1) e.g. in 2. slot of 9 slots but it should respond not until all routing is finished otherwise a collision may occur. Waiting can be ensured by a simple delay loop. Delay time is: number of hops left x time slot duration (`RTTSLOT x RTHOPS`). Before the loop `waitNewTick()` must be called first.

```
waitNewTick();
while (RTHOPS)          // RTHOPS - the rest of hops
{
  waitDelay(RTTSLOT); // RTTSLOT - timeslot
  RTHOPS--;
}
```

The answer can be sent not until this time is elapsed.
Additionally, this can be used for synchronization of all nodes in the network. E.g. a broadcast packet with a command to switch all lamps on in street lighting is received by individual lamps in different time slots. After this delay all lamps are synchronized and can be switched on at the same time.

The transmitting should be designed as follows:

```
...
setNodeMode();
RX = 0;                // To Coordinator
DLEN = 10;             // Data are already prepared in bufferRF

// for SFM
  RTDEF = 1;           // SFM
  getNetworkParams(); // Returns logical Node address in param2
  RTHOPS = param2;

// for DFM:
  RTDEF = 2;           // DFM
  RTHOPS = 0xFF;       // OS includes VRN in the packet into RTHOPS

  RTTSLOT = 2;         // 20 ms is enough for DLEN=10, see the formula on page 40.
  RFTXpacket();
  ...
```

Besides of the RTDEF register, the main difference between SFM and DFM is in setting of number of hops (`RTHOPS`):
- For SFM: `RTHOPS` = one's own logic address
- For DFM: `RTHOPS` = one's own VRN

Routing works in similar way like requests from Coordinator but time slot order is automatically reversed. Thus, for DFM answer from N1 (VRN=6) routing vector 5, 4, 3, 2, 1 is used:

- N1 (VRN 6) sends the answer, received by N2, N9, N4 and N5.
- N2, N4 and N5 do not resend the packet because their VRNs (7, 8, 9) are higher than sender's VRN (6).
- N9 (VRN = 4) waits one time slot and then resend the packet. This skipped slot should belong to N7 (VRN = 5), but it has not received the packet.
- The packet from N9 is received by N6, N3, N1 and N2.
- N1 and N2 do not respond because their VRNs (6 and 7) are higher than sender's VRN (4).
- N3 (VRN = 1) waits 2 slots
- N6 (VRN = 3) resends the packet in the next slot
- etc.

Thus, the Coordinator receives the packet twice:
- In penult slot from N8 (VRN = 2)
- In last slot from N3 (VRN = 1)

Note:
If a sender of DFM answer is not discovered (having no VRN) OS automatically ensures sending to the Node the request has been received from. Then routing normally goes on.

Appendix 1 – Memory maps

RAM map (PIC16LF1983)

Structure of each bank

Address offset	Memory region
00 0B	Core
0C 1F	SFR
20 6F	GPR
70 7F	Common

Common registers

Addr	Register	Access	Description
70	userReg0	R/W	User
71	userReg1	R/W	User
72	RFmodeByte	R	Current RF mode (set by setRFmode)
73	param2	R/W	Parameter for OS functions
74 75	param3	R/W	Parameter for OS functions
76 77	param4	R/W	Parameter for OS functions
78		–	
79		–	
7A		–	
7B		–	
7C		–	
7D		–	
7E		–	
7F	userInterface	R/W	Some information about current system, RF and network parameters

R/W – *read/write* R – *read only*

User registers

Bank	Address		Space
	Linear	Traditional	
11	2390	5C0	user 48 B
	23BF	5EF	
12	23C0	620	user 48 B
	23EF	64F	

Communication buffers

Bank	Address		Buffer
	Direct	Linear	
6	320	21E0	bufferINFO 64 B
	35F	221F	

Bank	Address		Buffer
	Direct	Linear	
7	3A0	2230	bufferCOM 64 B
	3DF	226F	

Bank	Address		Buffer
	Direct	Linear	
8	420	2280	bufferAUX 64 B
	45F	22BF	

Bank	Address		Buffer
	Direct	Linear	
9	4A0	22D0	bufferRF 128 B
	4EF	231F	
10	520	2320	
	54F	234F	

OS, RF and network parameters

Addr	Register	User access	Relates to	Description
042	sysReg2	R/W	some control flags	
1EC	XLPticks	R/W	RFRXpacket	Number of ticks to wait if _XLPnoWait is used
1ED	memoryLimit	R/W	Buffers	Number of bytes for buffer copying
4C0	DataOutBeforeResponseFRC	R	FRC	Delivered from Coordinator during FRC
5A0	ntwADDR	R	Bonding	Logical Node address
5A1	ntwVRN	R	Discovery	VRN
5A2	ntwZIN	R	Discovery	Zone index (Zone number + 1) E.g. for nodes in direct range from coordinator ntwZIN==1
5A3	ntwDID	R	Discovery	Discovery ID
5A4	ntwPVRN	R	Discovery	Parent VRN
5A5	ntwUSERADDRESS	R	setUserAddress	2 B user address
5A6				
5A7	ntwID	R	Coordinator ID	Network identification – NID0
5A8		R		Network identification – NID1
5A9	ntwVRNFNZ	R	Discovery	VRN of first Node in given zone
5AA	ntwCFG	R	setRoutingon/off,...	Network configuration
5AB	memoryOffsetFrom	R/W	Buffers	Offsets for buffer copy functions. See IQRF reference guide.
5AC	memoryOffsetTo	R/W		
5AD	userStatus	R/W	User variable	Cleared after power-on reset, unchanged after other resets
5AE	toutRF	R/W	RFRXpacket	Timeot for RFRXpacket() duration
5AF	RFspeed	R	setRFspeed	Current RF speed
5B0	RFpower	R	setRFpower	Current RF power
5B1	RFchannel	R	setRFchannel	Current RF channel
5B2	SPIpacketLength	R	SPI master	SPI packet length
5B6	lastRSSI	R/W	RFRXpacket	RSSI of last receipt
5B7	configFRC	R/W	FRC	FRC configuration
5B8	responseFRCvalue	R/W	FRC	To be returned to Coordinator during FRC response
5B9	DataInSendFRC	R/W	FRC	To be delivered to all Nodes during FRC
5BA				
5BB	sysReg1	R/W	some control flags	

Network INFO

Address	Register	Access	Description	
2A0	PIN	R/W	Packet information	
2A1	DLEN	R/W	Data length	
2A2		–		
2A3	RX	R/W	Addressee	
2A4	TX	R	Sender	
2A5		–		
2A6		–		
2A7	PID	R/W	Packet identification	
2A8	RTOTX	–	*Dedicated to OS*	
2A9	RTDEF	R/W	Routing algorithm	
2AA	RTHOPS	R/W	Number of hops per packet	Routing data
2AB	RTSLOT	R/W	Time slot duration	
2AC	RTDID	R/W	*Dedicated to OS*	
2AD	RTAUX	R/W	*Dedicated to OS*	
2AE	PNUM	–	*Dedicated to DPA*	
2AF	PCMD	–		
2B0	PPAR	–		

R/W – *read/write* R – *read only*

EEPROM map (inside the MCU, PIC16LF1983)

00	Available for Node only. Do not use for Coordinator.	Number of bonded Nodes	40	Available for Node only. Do not use for Coordinator.		80	Available for Node only. Do not use for Coordinator.		C0	Reserved by operating system. Do not use at all.	
01			41			81			C1		
02			42			82			C2		
03			43			83			C3		
04			44			84			C4		
05			45			85			C5		
06			46			86			C6		
07			47			87			C7		
08			48			88			C8		
09			49			89			C9		
0A			4A			8A			CA		
0B			4B			8B			CB		
0C			4C			8C			CC		
0D			4D			8D			CD		
0E			4E			8E			CE		
0F			4F			8F			CF		
10			50			90			D0		
11			51			91			D1		
12			52			92			D2		
13			53			93			D3		
14			54			94			D4		
15			55			95			D5		
16			56			96			D6		
17			57			97			D7		
18			58			98			D8		
19			59			99			D9		
1A			5A			9A			DA		
1B			5B			9B			DB		
1C			5C			9C			DC		
1D			5D			9D			DD		
1E			5E			9E			DE		
1F			5F			9F			DF		
20			60			A0	Application Info, 32 B		E0		
21			61			A1			E1		
22			62			A2			E2		
23			63			A3			E3		
24			64			A4			E4		
25			65			A5			E5		
26			66			A6			E6		
27			67			A7			E7		
28			68			A8			E8		
29			69			A9			E9		
2A			6A			AA			EA		
2B			6B			AB			EB		
2C			6C			AC			EC		
2D			6D			AD			ED		
2E			6E			AE			EE		
2F			6F			AF			EF		
30			70			B0			F0		
31			71			B1			F1		
32			72			B2			F2		
33			73			B3			F3		
34			74			B4			F4		
35			75			B5			F5		
36			76			B6			F6		
37			77			B7			F7		
38			78			B8			F8		
39			79			B9			F9		
3A			7A			BA			FA		
3B			7B			BB			FB		
3C			7C			BC			FC		
3D			7D			BD			FD		
3E			7E			BE			FE		
3F			7F			BF			FF		

Flash map

Standard	Extended	Configuration	OS
			0000
			.
			2BFF
	2C00		
	.		
	37BF		
		37C0	
		.	
		37DF	
			37E0
			.
			39FF
3A00			
.			
3FFF			

Appendix 2 – Channel maps

Caution: For channel selecting, users have to ensure observing local provisions and restrictions.

433 MHz band channel map

Channel	Center frequency [MHz]
0	433.10
1	433.20
2	433.30
Service channel 1	433.35
3	433.40
4	433.50
5	433.60
6	433.70
7	433.80
Service channel 2	433.85
8	**433.90**
9	434.00
10	434.10
11	434.20
12	434.30
13	434.40
Service channel 3	434.45
14	434.50
15	434.60
16	434.70

Default operational channel: 8

868 MHz band channel map for TR-72D, TR-75D, TR-76D and TR-78D

Channel	Center frequency [MHz]
0	863.15
1	863.25
2	863.35
3	863.45
4	863.55
5	863.65
6	863.75
7	863.85
8	863.95
9	864.05
10	864.15
11	864.25
12	864.35
13	864.45
Service channel 1	864.50
14	864.55
15	864.65
16	864.75
17	864.85
18	864.95
19	865.05
20	865.15
21	865.25
22	865.35
23	865.45
24	865.55
25	865.65
26	865.75
27	865.85
28	865.95
29	866.05
30	866.15
31	866.25
Service channel 2	866.30
32	866.35
33	866.45
34	866.55

Channel	Center frequency [MHz]
35	866.65
36	866.75
37	866.85
38	866.95
39	867.05
40	867.15
41	867.25
42	867.35
43	867.45
44	867.55
45	867.65
46	867.75
47	867.85
48	867.95
49	868.05
50	868.15
51	868.25
52	**868.35**
53	868.45
54	868.55
55	868.65
56	868.75
57	868.85
Service channel 3	868.90
58	868.95
59	869.05
60	869.15
61	869.25
62	869.35
63	869.45
64	869.55
65	869.65
66	869.75
67	869.85

Subband	Frequencies [MHz]	Duty (per hour)
h1.1	863.0 – 870.0	≤ 0.1 %
h1.2	863.0 – 870.0	≤ 0.1 %
h1.3	863.0 – 870.0	≤ 0.1 %
h1.4	868.0 – 868.6	≤ 1.0 %
h1.5	868.7 – 869.2	≤ 0.1 %
h1.6	869.4 – 869.65	≤ 10.0 %
h1.7	869.7 – 870.0	≤ 1.0 %

The table of subbands above is just intended as an example valid for EU. The user have to ensure full observing all local provisions and restrictions including subsequent amendments.

E.g., in EU, channels 61 to 67 are recommended exclusively for alarm systems including social alarms and alarms for security and safety.

Default operational channel: 52

868 MHz band channel map for TR-77D

Channel	Center frequency [MHz]
45	867.65
46	867.75
47	867.85
Service channel 1	867.90
48	867.95
49	868.05
50	868.15
51	868.25
52	868.35
53	868.45
54	868.55
55	868.65
Service channel 2	868.70
56	868.75
57	868.85
58	868.95
59	869.05
60	869.15
Service channel 3	869.20
61	869.25
62	869.35
63	869.45
64	869.55
65	869.65
66	869.75
67	869.85

Subband	Frequencies [MHz]	Duty (per hour)
h1.1	863.0 – 870.0	≤ 0.1 %
h1.2	863.0 – 870.0	≤ 0.1 %
h1.3	863.0 – 870.0	≤ 0.1 %
h1.4	868.0 – 868.6	≤ 1.0 %
h1.5	868.7 – 869.2	≤ 0.1 %
h1.6	869.4 – 869.65	≤ 10.0 %
h1.7	869.7 – 870.0	≤ 1.0 %

The table of subbands above is just intended as an example valid for EU. The user have to ensure full observing all local provisions and restrictions including subsequent amendments.

E.g., in EU, channels 61 to 67 are recommended exclusively for alarm systems including social alarms and alarms for security and safety.

Default operational channel: 52

916 MHz band channel map for TR-7xD

Channel	Center frequency [MHz]
0	902.25
1	902.35
2	902.45
3	902.55
4	902.65
5	902.75
6	902.85
7	902.95
8	903.05
9	903.15
10	903.25
11	903.35
12	903.45
13	903.55
Service channel 1	903.60
14	903.65
15	903.75
16	903.85
17	903.95
18	904.05
19	904.15
20	904.25
21	904.35
22	904.45
23	904.55
24	904.65
25	904.75
26	904.85
27	904.95
28	905.05
29	905.15
30	905.25
31	905.35
Service channel 2	905.40
32	905.45
33	905.55
34	905.65
35	905.75
36	905.85
37	905.95
38	906.05
39	906.15
40	906.25
41	906.35
42	906.45
43	906.55
44	906.65
45	906.75
46	906.85
47	906.95
48	907.05
49	907.15
50	907.25
51	907.35
52	907.45
53	907.55
54	907.65
55	907.75
56	907.85
57	907.95
Service channel 3	908.00
58	908.05
59	908.15
60	908.25
61	908.35
62	908.45
63	908.55
64	908.65
65	908.75
66	908.85
67	908.95
68	909.05
69	909.15
70	909.25
71	909.35
72	909.45
73	909.55
74	909.65
75	909.75
76	909.85
77	909.95
78	910.05
79	910.15
80	910.25
81	910.35
82	910.45
83	910.55
84	910.65
85	910.75
86	910.85
87	910.95
88	911.05
89	911.15
90	911.25
91	911.35
92	911.45
93	911.55
94	911.65
95	911.75
96	911.85
97	911.95
98	912.05
99	912.15
100	912.25
101	912.35
102	912.45
103	912.55
104	**912.65**
105	912.75
106	912.85
107	912.95
108	913.05
109	913.15
110	913.25
111	913.35
112	913.45
113	913.55
114	913.65
115	913.75
116	913.85
117	913.95
118	914.05
119	914.15
120	914.25
121	914.35
122	914.45
123	914.55
124	914.65
125	914.75
126	914.85
127	914.95
128	915.05
129	915.15
130	915.25
131	915.35
132	915.45
133	915.55
134	915.65
135	915.75
136	915.85
137	915.95
138	916.05
139	916.15
140	916.25
141	916.35
142	916.45
143	916.55
144	916.65
145	916.75
146	916.85
147	916.95
148	917.05
149	917.15
150	917.25
151	917.35
152	917.45
153	917.55
154	917.65
155	917.75
156	917.85
157	917.95
158	918.05
159	918.15
160	918.25
161	918.35
162	918.45
163	918.55
164	918.65
165	918.75
166	918.85
167	918.95
168	919.05
169	919.15
170	919.25
171	919.35
172	919.45
173	919.55
174	919.65
175	919.75
176	919.85
177	919.95
178	920.05
179	920.15
180	920.25
181	920.35
182	920.45
183	920.55
184	920.65
185	920.75
186	920.85
187	920.95
188	921.05
189	921.15
190	921.25
191	921.35
192	921.45
193	921.55
194	921.65
195	921.75
196	921.85
197	921.95
198	922.05
199	922.15
200	922.25
201	922.35
202	922.45
203	922.55
204	922.65
205	922.75
206	922.85
207	922.95
208	923.05
209	923.15
210	923.25
211	923.35
212	923.45
213	923.55
214	923.65
215	923.75
216	923.85
217	923.95
218	924.05
219	924.15
220	924.25
221	924.35
222	924.45
223	924.55
224	924.65
225	924.75
226	924.85
227	924.95
228	925.05
229	925.15
230	925.25
231	925.35
232	925.45
233	925.55
234	925.65
235	925.75
236	925.85
237	925.95
238	926.05
239	926.15
240	926.25
241	926.35
242	926.45
243	926.55
244	926.65
245	926.75
246	926.85
247	926.95
248	927.05
249	927.15
250	927.25
251	927.35
252	927.45
253	927.55
254	927.65
255	927.75

Default operational channel: 104

916 MHz band channel map for TR-7xD-IL

Due to local government regulation, transceivers to be operated in Israel are distributed with limitation for 916 MHz band and channels from 133 to 140 only. Such transceivers (e.g. TR-72D-IL) are identified by the IL postfix in IQRF IDE TR Module Info.

Channel	Center frequency [MHz]
133	915.55
134	915.65
Service channel 1	915.70
135	915.75
136	915.85
Service channel 2	915.90
137	915.95
138	**916.05**
139	916.15
Service channel 3	916.20
140	916.25

Default operational channel: 138

Appendix 3 – RFPGM - RF Programming™

Standard version

IQRF TR modules can be uploaded even in a wireless way (RFPGM – RF Programming™). During RF upload the programmed TR is not inserted in the programmer but it is connected to the programmer in a wireless way by an auxiliary TR module.

RFPGM allows:
- Upload TRs also in final application boards (housed, soldered etc.)
- More TRs uploaded simultaneously (e.g. all IQMESH Nodes in one stroke)

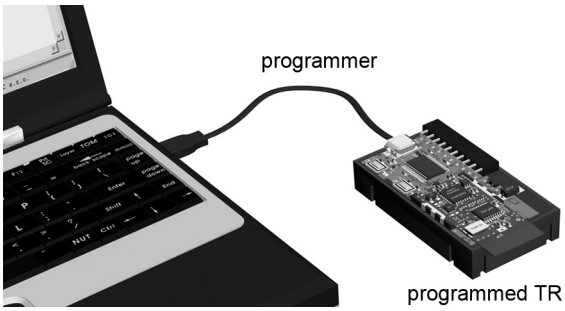


Fig. 1: Standard (wired) upload in a programmer

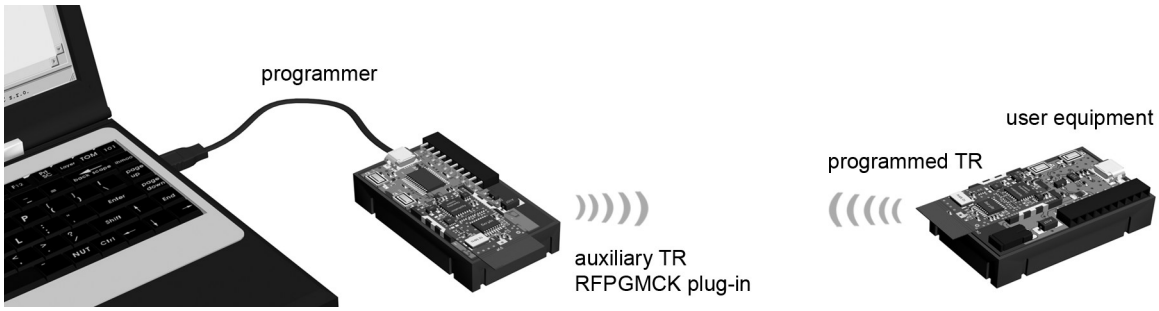


Fig. 2: Wireless upload (RFPGM)

The auxiliary TR module has to be configured as RF programmer by IQRF IDE using the *Create CATS* function (see the Example below).

Note:

RF programmer must be created from the TR module having the proper RF band (in advance).

To utilize RFPGM, the *RF Programming* checkbox must be active in IQRF IDE. During wireless upload the programmed TR module is not in the standard programming mode but in the RFPGM one. The standard mode is emulated by the auxiliary TR module in the programmer. From the IQRF IDE point of view all is the same as for standard programming except of the fact that the *TR Module Info* window and the *Reset TR Module* button in IQRF IDE relates to the auxiliary TR module.

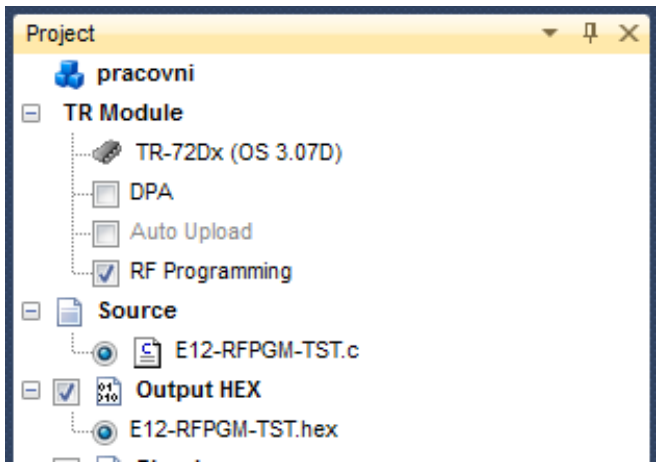


Fig.3: Project window

RFPGM features

- Single / Dual channel programming. RF channels can be selected, see chapters TR module configuration (above) and RF programmer configuration (below). For LP RFPGM just a Single channel programming is available.
- Fixed RF bit rate (19.8 kb/s).
- "Non-networking" communication based on system broadcast packets (no addressing, unidirectional, no answers returned). This allows to upload a lot of TRs at the same time.
- Several RFPGM parameters can be configured.
- Files can be uploaded as follows:
  - Unencrypted (`.HEX`) and encrypted (`.IQRF`) files into memories inside the MCU:
    - Flash as well as EEPROM: in full range
    - Serial EEPROM: the lowest 2 kB only
  - TR configuration files (`.XML`), except of RF band and RFPGM configuration.
- All RFPGM packets are checked. If not all packets are received properly, the application program is not launched and TR module stays still in RFPGM mode (even after reset). This state is indicated by the *Incorrect upload* bit when the configuration is read out from the TR module. See chapter *TR configuration* for more information.
- Due to increased robustness, the relative RF range is lower for about 35 %.
- The communication robustness can further be adjusted by configuration of TR programmer, see below. It can be specified whether one or two channels will be used. If dual channel is selected, all packets are sent via both channels A and B, otherwise just the channel A is used. Moreover, every RFPGM packet can be sent more times. Number of repeats is also selectable. Thus, RFPGM features can be adapted to specified environment (with respect to the noise level) to achieve optimal throughput and reliability.
- RFPGM functions implemented in OS (see the IQRF Reference guide):
  - `runRFPGM()` – switch TR to the RFPGM mode
  - `enableRFPGM()` – request to enable switching TR to the RFPGM mode after every reset
  - `disableRFPGM()` – request to disable switching TR to the RFPGM mode after every reset (factory default)
  - `setupRFPGM(x)` – setup RFPGM parameters
- RFPGM mode can be entered by:
  - Reset (if enabled)
  - `runRFPGM()` function (unconditionally)
- RFPGM mode can be terminated by:
  - Low level on dedicated pin (see the TR module datasheet) for at least ~0.25 s (single channel) or ~0.5 s (dual channel) (if enabled). This time must be prolonged up to 2 s in case of strong RF noise.
  - ~1 minute after reset (if enabled and if RFPGM is not in progress)
  - By the *End RF Programming Mode* button from IQRF IDE (unconditionally).

RF programmer configuration

After creating or connecting an RF programmer, IQRF IDE automatically activates the *RF Programming* checkbox and displays the window for selecting parameters of the RF programmer. These parameters correspond to setting in window Documents → Project Properties → Programming → RF Programmer Parameters. That is why it is possible to have parameters selected in advance and stored in the Project. See IQRF Help for details.

RF Band

RF band should be the same for the programmer and for all programmed TR modules. If the programmer has a different band, IQRF IDE issues a warning.
If there is a need to change RF band (applicable for TR modules allowing more bands only) of an existing RF programmer, the following steps should be performed:

- Remove RF programmer
- Change the band of TR module via TR Configuration
- Create RF programmer

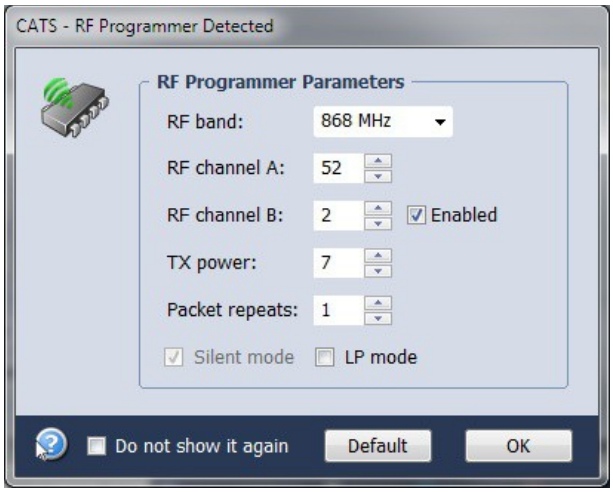


Fig. 4: RFPGM parameters

RF Channel A

RF channel A selection for given RF band.

RF Channel B

RF channel B selection for given RF band.
Not available for LP RFPGM.

Enabled

Enables channel B – Single / Dual channel selection.

TX Power

RF output power setting.

Packet repeat

Number of packet repetitions on each RF channel.

Silent mode

This selection has no meaning for TR-7xD. RFPGM uses system packets only.

LP mode

Activate when uploaded TR modules use low power RFPGM mode.

LED indication

Operation		Auxiliary TR		Programmed TR	
		Green	Red	Green	Red
RFPGM mode	Standard	short flash in 1 s period	–	continuous	short flashes in 2 s period
	LP			4 x per second, weakly	–
Uploading		short flash in 1 s period	fast flashing (in LP a bit slower)	continuous	fast flashing
RFPGM mode finished		–	–	–	single flash for 1.5 s

RFPGM usage

Typical requirement used to be a different behavior during development (when modified program is uploaded repeatedly) and after finishing the development (when following-up reprogramming is not desired at all or is required for service purposes on special request only).

The `enableRFPGM()` is very important during development. Special care should especially be taken for applications with TR modules inaccessible for wired upload. E.g. an error in user program under development can cause that the `runRFPGM()` function can not be accessed any more. If the TR can not be reprogrammed externally, the only way how to force programming mode is having a possibility to switch to RFPGM automatically after reset. Thus, main purposes of `enableRFPGM()` are:

- To enable entering RFPGM while routine development in a fast and reliable way.
- A loop-hole against user's errors during development.

But it is recommended to keep a way of how to invoke RFPGM mode even in completed and debugged designs on request (at least to allow services and future upgrades). That is why `disableRFPGM()` should not be used unless there is a user's defined way of how to invoke back the RFPGM mode via `runRFPGM()` or `enableRFPGM()`. It is up to the user to define the entry point into RFPGM, e.g. using a jumper, pushbutton or a special RF packet. If being omitted there is no any other way of getting back into RFPGM unless removing the module from a device and placing it into the programmer.

Thus, RF upload is fully under the user's control. Invoking the RFPGM mode, its termination, behavior after reset – everything can be adapted according to the user's needs.

Example

Wireless upload procedure for SIM card compatible TR modules can be tested using the E12-RFPGM-TST demo program:

1. Insert the programmed TR to the programmer and upload (not in a wireless way) the E12-RFPGM-TST demo program.
2. Remove the TR module from the programmer and insert it in the DK-EVAL-04x kit. Running application is indicated by red LED flashing. It is possible to switch the TR module in RFPGM mode anytime by pressing the SW1 pushbutton (near the USB connector). For LED indication see the table above.
3. Plug another TR module in the programmer (or use the GW-USB-06 gateway) and create an RF programmer in IQRF IDE (*Tools - CATS Service Tools - Control* menu, *Create CATS* button). The CATS plug-in is uploaded (still not in a wireless way).
4. Arrange both TR modules according to Fig. 2 for RFPGM upload. Once the programmer is connected to IQRF IDE, the window for setting of parameters (Fig. 4) is opened and the *RF Programming* checkbox is automatically activated. Now select requested parameters.
5. For test purpose change LED flashing from the red LED to the green one in E12-RFPGM-TST demo program and compile this.
6. Switch the programmed TR module in RFPGM mode by pressing the SW1 pushbutton (DK-EVAL-04x) and upload the modified application by IQRF IDE. After RFPGM finishing the red LED flashes for 1.5 s and then the application starts up automatically. Now the green LED should flash instead of the red one which indicates that the application has been changed. (RFPGM mode can be terminated without RF uploading by the SW1 pushbutton pressing for ~0.7 s.)

Removing the RGPGM programmer

The RFPGM programmer can be removed from the TR module in IQRF IDE by following ways:

- By the *Remove CATS* button at tool *CATS Service Tools → Control*.
- By uploading another application into the TR module. However it can not be done just by the Upload key (*F5)* but the *Enter Programming Mode* (F6) button must be clicked first.

TR modules are delivered with RFPGM after reset disabled. Versions with RFPGM after reset enabled are available on request.

Appendix 4 – Coordinator Bonding and Discovery Data

The Coordinator stores the results of bonding and discovery processes into internal and external EEPROM memories. The data stored there can be used to optimize a network traffic, to visualize a discovered network paths (the paths which individual Nodes has been discovered by, not the real network topology with all routing paths), to make an inventory, etc.

All described memory blocks are to be read only. `clearAllBonds()` must be used to initialize serial EEPROM before creating the IQMESH network (before the first bonding). Several OS functions (e.g. `bondNewNode`, `nodeAuthorization`, `discovery` and `sendFRC`) check serial EEPROM content and immediately returns an error if a non-consistency is found.

Bonded devices count

The byte holding the number of bonded devices can be read from the internal MCU EEPROM at address `0x00`.

Bonded devices bitmap

The bitmap of all bonded devices (with logical addresses 1 to 239) is a 30 byte long block stored at internal MCU EEPROM from address `0x01`. Node N1 (the node having logical address 1) is represented by bit 1 of the 1st byte of the block, N2 by bit 2 of the 1st byte, …, N8 by the bit 0 of the 2nd byte, …, N239 by the bit 7 of the 30th byte.

Address	**0x01**								**0x02**								**...**	**0x1E**							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0
Node	N7	N6	N5	N4	N3	N2	N1	n/a	N15	N14	N13	N12	N11	N10	N9	N8		N239	N238	N237	N236	N235	N234	N233	N232

Discovered devices bitmap

The bitmap of all discovered devices is a 30 byte long block stored at internal MCU EEPROM from address `0x20`. The mapping format of the nodes is the same as at Bonded devices bitmap except the starting address is `0x20`.

Address	**0x20**								**0x21**								**...**	**0x3D**							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0
Node	N7	N6	N5	N4	N3	N2	N1	n/a	N15	N14	N13	N12	N11	N10	N9	N8		N239	N238	N237	N236	N235	N234	N233	N232

MID array

The MID array stores the MID (4 B) of each bonded Node. The array is 1920 bytes long and can be found at external EEPROM starting from address `0x4000`. Complete 4 B of MID are stored, LSB first, followed by 4 B reserve for each Node. The MID record is not cleared after unbonding a Node.

Address	**0x4000 – 0x4007**	**0x4008 – 0x400F**		**0x0004 – 0x0005**		**...**	**0x4778 – 0x477F**	
Node	–	N1		N2		...	N239	
Node's MID [4 B]	n/a	MID[0-3]	Reserved	MID[0-3]	Reserved	...	MID[0-3]	Reserved

VRN array

VRN array stores VRN of every discovered Node. The array is 240 bytes long and can be found at serial EEPROM starting from address `0x5000`. The array is indexed by Node's logical address. If a Node is not discovered then stored VRN equals to 0.

Address	**0x5000**	**0x5001**	**0x502**	**...**	**0x50EF**
Node's VRN	*n/a*	N1	N2	...	N239

Zones array

Zones array stores zone of every discovered node. The array is 240 bytes long and can be found at serial EEPROM starting from address `0x5200`. The array is indexed by node's logical address. If node is discovered then stored Zone value equals to the actual Zone+1, otherwise it equals to 0.

Address	**0x5200**	**0x5201**	**0x5202**	**...**	**0x52EF**
Node's zone	*n/a*	N1	N2	...	N239

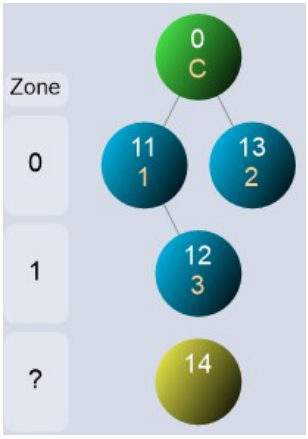
Parents array

Parents array stores the parent's logical address of every discovered node. The array is 240 bytes long and can be found at serial EEPROM starting from address `0x5300`. The array is indexed by node's logical address. If a Node is not discovered then stored parent's logical address is not defined.

Address	0x5300	0x5301	0x5302	...	0x53EF
Node's parent address	*n/a*	N1	N2	...	N239

Example

The following example shows the memory contents for a simple network comprising of 4 Nodes (logical address 11-14). Nodes 11-13 are discovered, Node 14 is not discovered. Nodes 11+13 are located at Zone 0, node 12 is at Zone 1:



Network Information					
Address	MID	Parent Addr	VRN	Zone	Discovered
0	810002C2	-	-	-	-
11	810002C7	0	1	0	✔
12	810002C4	11	3	1	✔
13	810002C3	0	2	0	✔
14	810002C6	-	-	-	-

Internal EEPROM

Address	0x00	0x01	0x02	...	0x20	0x21	...	0x3D
Value	0x04	0x00	0x78	0x00	0x00	0x38	0x00	0x00
Description	Bonded Nodes count	Not bonded Node	N11-N14 are bonded	Not bonded Nodes	Not discovered Node	N11-N13 are discovered	Not discovered Nodes	Not discovered Node

Serial EEPROM

Address	0x5000-0x500A	0x500B	0x500C	0x500D	0x500E	0x500F-0x50EF
Node's VRN	0x00	0x01	0x03	0x02	0x00	0x00
Description	not discovered nodes	N11 VRN	N12 VRN	N13 VRN	N14 VRN (not discovered)	not discovered nodes

Address	0x5200-0x520A	0x520B	0x520C	0x520D	0x520E	0x520F-0x52EF
Node's Zone	0x00	0x01	0x02	0x01	0x00	0x00
Description	not discovered nodes	N11 zone	N12 zone	N13 zone	N14 zone (not discovered)	not discovered nodes

Address	0x5300-0x530A	0x530B	0x530C	0x530D	0x530E	0x530F-0x53EF
Node's parent	n/a	0x00	0x0B	0x00	n/a	n/a
Description	not discovered nodes	N11 parent	N12 parent	N13 parent	N14 parent (not discovered)	not discovered nodes

Appendix 5 – Migration from lower OS versions

Migration from OS v4.02D to v4.03D

Features	OS v4.02D	OS v4.03D
Direct LED switching on (`_LEDR = 1`, `_LEDG = 1`)	Available	Possible but unnecessary. Use LED control OS functions, e.g. `setLEDR()` and `setLEDG()` instead.
`eeeReadData()` and `eeeWriteData()` default block length	16 B	64 B
`eeeWriteData()` 64 B blocks boundary crossing	Not allowed	Allowed
~~Side effect "`bufferAUX` modified" by functions `bondNewNode()`, `prebondNodeAtCoordinator()` and `nodeAuthorization()`~~	~~No~~	~~Yes~~
Service RF channels	Not available	Fixed 3 dedicated channels are used for bonding and Smart connect.
RF channels for RFPGM and DPA DSM	The same as the operational channels	Shifted by 50 kHz above the operational channels
Operating RF channel of the Node to be bonded to the network	Must be specified in TR configuration of given Node.	Need not be specified. Inherited from the Coordinator. Stored to Node's Configuration during bonding.
Standard FRC	Available	Not available. Use Advanced FRC.
4 B FRC	Not available	Available
Location of User key for User encryption/decryption	Specified by `setUserKey()`	Specified by `setUserKey()` or placed in `bufferINFO`

Variables for bidirectional user data exchange during (pre)bonding	`userBondingData[4]`	`hostUserDataToSend[4]`
		`nodeUserDataToSend[4]`
		`hostUserDataReceived[4]`
		`nodeUserDataReceived[4]`

IBK and Smart connect	Not implemented	Implemented
IBK available via `moduleInfo()`	No	Optionally

Migration from OS v4.00D to v4.02D

Features	OS v4.00D	OS v4.02D
TR and DCTR transceiver types differentiation	Differentiated	Not differentiated. Even a transceiver named TR features both OS and DPA (data controlled) approaches.
`getTemperature()` return value	`uns8`	`int8`
`eeReadData()` return value	Not documented	Documented
TR upload (see IQRF SPI Technical guide)		Extended possibilities
Immunity against possible failure of RF IC		Extended
Bug in 2B FRC individual packets processing for not discovered Nodes in range with the Coordinator only.	Yes	Fixed

Migration from OS v4.00D to v4.01D

Do not use OS v4.01D at all due to a serious bug in RFPGM (wireless upload).

Migration from OS v3.08D to v4.00D

Features		OS v3.08D	OS v4.00D
RF packet interoperability between OS v3.xxD and v4.xxD		Not interoperable	
Networking with more than 240 devices		Supported	Not supported
IQMESH implementation		DPA or non-DPA	DPA only
`setUserAddress()`		Implemented	Not implemented
`setRFband()`		Implemented	Not implemented
Networking RF packet propagation time			Slightly prolonged due to encryption, see page 34
RF IC restart after RF failure detection		Not available	`wasRFICrestarted()`
`deepSleep()`		Not implemented	Implemented
`getINDF0()` and `getINDF1()`		Obsolete	Not implemented
`clearBufferINFO()` and `clearBufferRF()`		Complete buffer is cleared	Can be limited by `memoryLimit`
`swapBufferINFO()`		Complete buffer is swapped	Can be limited by `memoryLimit`
`eeReadData()` return value		None (void)	Bit (at least one zero read)
`eeReadData()` and `eeWriteData()` data length		1 B to 32 B, default 32 B	1 B to 64 B, default 32 B
Using `memoryOffsetTo` and `memoryOffsetFrom` with `eeReadData()` and `eeWriteData()`		Not available	Available
Serial EEPROM mapping	Dedicated to OS	0x0000 – 0x07FF, default cleared, not write protected	0x4000 – 0x7FFF, default not cleared, write protected
	User available	0x0800 – 0x7FFF	0x0000 – 0x3FFF
Application info default setting		`"Hello everybody. IQRF is here!"`	Cleared
`writeToRAM()` and `appInfo()` implementations		OS functions	Macros
Preamble length in STD TX		3 ms	Selectable 4 ms or 8 ms
Preamble quality check for checkRF(x)		Not available	Available
Range of `checkRF(level)` parameter		0 – 100	0 – 90
Network variables for DPA renamed		`MPRW0`, `MPRW1`, `MPRW2`	`PNUM`, `PCMD`, `PPAR`
Alternative names of routing variables		`RTDT0-3` also allowed	`RTHOPS`, `RTSLOT`, `RTDID`, `RTAUX` only
Alternative names for `RTAUX` (formerly `RTDO3`)		`RTDT3`, `RTV3`, `RX2BH`, `OTX2BH`	`RTAUX` only
Number of Nodes addressed by FRC (bytes collected)		62 (1 B mode), 30 (2 B mode)	63 (1 B mode), 31 (2 B mode)
FRC duration			Slightly longer
`bondRequest()`		Obsolete	Not implemented
Possible vacations within bonding addresses during `bondNewNode(0)` and `nodeAuthorization(x)`		Should be avoided	Allowed, the lowest free address is used (unless otherwise requested)
Parameter 0xFF at `nodeAuthorization(x)`		Not available	Available
Node prebonding functions (see IQRF OS Reference guides for differences)		`prebondNode()`	`prebondNodeAtNode()` `prebondNodeAtCoordinator()`
Flag `_prebondNode` for Node prebonding and authorization		Must be handled by the user	Not used
User data exchanged during prebonding		2 B, via special variables	4 B, via `bufferRF`
Bonding variables `DataInPrebondNode`, `DataOutPrebondNode`, `DataInBondRequestAdvanced`, `DataOutBondRequestAdvanced`		Used	Replace by universal variable `UserBondingData`
MID passed by `nodeAuthorization()`		Lower 2 B, via `MIDoutBondRequest`	Complete 4 B, via `BondingNodeMID`
Waiting for `nodeAuthorization` finishing		No waiting (non-blocking, running in OS background)	Waiting (blocking, running in OS foreground). No user delay is needed.
Networking packets allowed	RX	Allowed	Node authorization system packets only

for prebonded Nodes	TX	Allowed	Not allowed
`amIRecipientOfFRC()` return value for non-selective FRC		Always `true`	`true` only for Nodes with addresses within given range
RF communication security / encryption		Proprietary coding	AES-128 (User, Networking and Access)
TRs to be operated in Israel (IL version)		Not available	Available
Basic IQRF header file		`template-basic.h`	Renamed to `IQRF.h`

Documentation and information

1 **IQRF OS Reference guide**
2 **Memory maps**: this document, Appendix [1]
3 **IQRF home page**
4 **IQMESH specification**
5 **SPI specification**
6 **IQRF support**
7 **TR-72D datasheet**
TR-75D datasheet
TR-76D datasheet
TR-77D datasheet
TR-78D datasheet
8 **PIC16LF1938 datasheet**
9 **IQRF IDE**
10 **Examples** (included in the StartUp Package)
11 **IQRF Quick start guide**
12 **IQMESH, Technology for Wireless Mesh Networks** article

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

Document revision

- 181025 First release for IQRF OS v4.03D.

Sales and Service

Corporate office

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

Technology and development

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

Partners and distribution

www.iqrf.org/partners

Quality management

ISO 9001 : 2009 certified

Trademarks

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

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 IQRF Tech 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).

On-line support: support@iqrf.org

Smarter Wireless. Simply.