# IQRF DPA Framework

## Technical guide

### Version v2.01

### For IQRF OS v3.05D

# Contents

# 1 Introduction

DPA protocol is a simple byte oriented protocol used to control services and peripherals of IQMESH network devices (coordinator and nodes) by SPI or UART interfaces.

DPA protocol implementation is distributed in the form of IQRF plug-in. Full version runs only at IQRF Data Controlled Transceivers (DCTR). There is a demo version that can run at ordinary - IQRF Smart Transceivers (TR).

The demo version has the following features:

- Maximum node network address is 5. Demo node device having unsupported address flashes periodically red LED after reset. Demo coordinator does not allow to address, to bond and to rebond node with an unsupported address.
- Some Custom DPA Handler events are not raised at demo version.
- Only one user peripheral PNum = 0x20 with one PCmd = 0x00 is supported.
- Discovery and FRC processes are indicated by LEDs flashing by default.

Please note that implementation of LP and XLP modes at DPA is not released yet and any information concerning these modes within the document is preliminary.

# 2 Basics

DPA protocol uses byte structured messages to communicate at IQMESH network. Every message always contains four mandatory parameters NAdr, PNum, PCmd and HwProfile (*foursome* from now). The message can optionally hold data (array of bytes often refered to as PData throughout the document) to be transmitted or received. They are always described next to the foursome throughout this document. Although foursome parameters are typically described next to each other in this document, they do not have to be stored at consecutive memory addresses at the real scenario. The same rule does not apply to the data in the message.

Please note that a response, confirmation and notification (with a small exception) DPA messages always contains the same NAdr, PNum and PCmd as the original request message except the response message is flagged by the most significant bit of PCmd.

## 2.1 Device types

There are several device types depending on what type of network device it implements. For each device type there is dedicated IQRF plug-in prepared for upload.

- [C] A “pure” IQMESH Coordinator device
- [N] A typical IQMESH Node device
- [NC] This device implements both IQMESH Node functionality in the main network as well as Coordinator functionality in the optional subordinate network.

## 2.2 RF Modes

There is a separate DPA implementation for each of the IQRF RF modes (STD, LP, XLP) (as well as for Device types) prepared in the form of IQRF plug-in. Currently actions bonding, remote bonding and discovery are supported at STD mode only.

## 2.3 Interfaces

The choosen interface transfers DPA message data to/from the connected device. Data consist of successively stored foursome and optional data. By default the distributed plug-in supports SPI interface. Plug-ins supporting UART interface have “IFACEUART” at their file name.

### 2.3.1 SPI

The SPI interface is implemented using IQRF SPI protocol described at document "SPI Implementation in IQRF TR modules". The document specifies how to setup SPI master and the communication over the SPI. The DPA protocol corresponds to the DM and DS bytes of IQRF SPI protocol.

### 2.3.2 UART

UART is configured 8 data bits, 1 stop bit, no parity bit. UART baud rate is stored at HWP Configuration.
HDLC byte stuffing protocol is used to transfer data. Every data frame starts and ends with byte 0x7e (Flag Sequence). When actual data byte equals to 0x7e (Flag Sequence) or 0x7d (Control Escape) then it is replaced by two bytes: 1st byte 0x7d (Control Escape) and 2nd byte equal to original byte value XORed by 0x20 (Escape Bit).

Additionally an 8-bit checksum is computed by XORing all original byte values and constant value 0x5F. The checksum is added after all data bytes and coded by the same byte stuffing algorithm.

**Example**

The example shows how 3 input bytes of data (0x01, 0x7d, 0x5d) including checksum 0x7e (0x5f ^ 0x01 ^ 0x7d ^ 0x5d) are coded using HDLC byte stuffing resulting in total output 8 bytes.

| Data in index | | 0 | 1 | | 2 | *checksum* | | |
|---|---|---|---|---|---|---|---|---|
| Data in | | 0x01 | 0x7d | | 0x5d | 0x7e | | |
| Data out index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Data out | **0x7e** | 0x01 | **0x7d** | 0x5d | 0x5d | **0x7d** | 0x5e | **0x7e** |

## 2.4 General message parameters

*All numbers are in hexadecimal form unless otherwise noted.*

| Parameter | Value [hex] | | Description |
|---|---|---|---|
| NAdr<br><br>[2B] | 00<br>01-EF<br>F0-FB<br>FC<br>FD-FE<br>FF<br>100-FFFF | IQMESH Coordinator<br>IQMESH Node address<br>Reserved<br>Local (over interface) device<br>Reserved<br>IQMESH Broadcast address<br>Reserved for 2 byte address | Network device address.<br>Although it is 2 bytes wide, the 2B addressing is not supported yet.<br>NAdr 2 bytes are coded using little-endian style. |
| PNum<br><br>[1B] | 00<br>01<br>02<br>03<br>04<br>05<br>06<br>07<br>08<br>09<br>0A<br>0B<br>0C<br>0D<br><br>20-6F<br><br>70-FF | COORDINATOR<br>NODE<br>OS<br>EEPROM<br>EEEPROM<br>RAM<br>LEDR<br>LEDG<br>SPI<br>IO<br>Thermometer<br>PWM *<br>UART<br>FRC<br><br>User peripherals<br><br>Reserved | Peripheral number<br>(0x00 – 0x1F reserved for standard peripherals)<br><br>1st user peripheral must be always 0x20, 2nd must be 0x21 etc. |
| PCmd<br><br>[1B] | 0-3E<br>3F | Reserved | Command specifying an action to be taken. Actual allowed value range depends on the peripheral type.<br>The most significant bit is reserved for indication of DPA response message. |
| HwProfile<br><br>[2B] | 0000<br>0001-0100<br>0101-F000<br>F001-FFFF | Default HW Profile<br>Reserved<br>Certified HW Profiles<br>Reserved | HW profile uniquely specifies the functionality of the device, the user peripherals it implements, its behaviour etc. Only device having the same HwProfile as the DPA request will execute the request. When 0xFFFF is specified then device with any HW profile will execure the request.<br>Note – actual HwProfile numbers used throughout this document are fictitious ones. |
| PData<br>[0-56B] | Array of bytes. The maximum length is limited.<br>In the current DPA version the limit is 56 bytes. | | Optional message data. |

** Available at Demo version and only in [N] device.*

## 2.5 Communication flow

DPA protocol (messages) is transferred over interface that connects TR module (“slave”) to a superordinate system (”master”).

- Master sends **DPA request**
- If addressee (NAdr) is a (remote) IQMESH Node, not a local over the interface connected device (applies only to coordinator)
  - Device immediately sends **DPA confirmation** back to the interface master
  - Node processes the DPA message
- If the DPA message does not have a read-only (can be configured by bit EnableSPInotificationOnRead) side-effect and the interface is configured for the DPA communication at the node side, then the node sends **DPA notification** to its SPI master
  - If the DPA message was not sent using broadcast address
    - Node returns **DPA response** back to coordinator via RF
    - Coordinator receives the **DPA response** and re-sends it to the interface master
- In case of a local device
    - Device processes the DPA message
    - Device returns **DPA response** back to interface master

### 2.5.1 DPA request

DPA request consists of *foursome* with optional data, depending on the actual request. DPA request is executed only if the specified HW profile matches the HW profile of the device unless HW profile in the foursome equals to 0xFFFF (`HWProfile_DoNotCheck`).

### 2.5.2 DPA confirmation

DPA confirmation confirms a reception of DPA request by interface slave to interface master. It consists of the same *foursome* that was part of the original DPA request plus following 5 additional bytes:

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| STATUS_CONFIRMATION | DPA Value | Hops | Timeslot length in 10 ms units | Hops Response |

DPA Value — See description.
Hops — Number of hops used to deliver the DPA request to the addressed node.
Timeslot length — Timeslot length used to deliver the DPA request to the addressed node. Please note that the timeslot used to deliver the response message from node to coordinator can have a different length.
Hops Response — Number of hops used to deliver the DPA response from the addressed node back to coordinator. In case of broadcast this parameter is 0 as there is no response.

IQMESH timeslot length depends on the data length within the DPA messages (the values may change depending on the version of the DPA protocol and IQRF OS version) and the RF mode (STD, LP, XLP) used:

| Data length [B] | | | Timeslot length [ms] | | |
|---|---|---|---|---|---|
| **STD** | **LP** | **XLP** | **STD** | **LP** | **XLP** |
| < 19 | < 19 | < 8 | 30 | 80 | 950 |
| 19 – 41 | 19 – 41 | 8 – 31 | 40 | 90 | 960 |
| > 41 | > 41 | 32 – 54 | 50 | 100 | 970 |
| | | > 54 | | | 980 |

This knowledge can be used to implement a precise timing of the control system (master) connected to the coordinator device by interface in order to prevent data collision (e.g. when another DPA request is sent to the network before a routing of the previous communication is finished) at the network.

1. Wait till the previous IQMESH routing is finished (see step 7)
2. Make sure the interface is ready (e.g. SPI status is `ReadyCommunication`) and no data remained for reading from interface.
3. Send DPA request via interface.
4. Receive DPA confirmation via interface. Remember the time when the confirmation was received (to be used later at step 7)
5. Now wait *Hops * Timeslot length * 10 ms* till the DPA Request routing is finished.
6. Read DPA response from the interface within the time *Hops Response * Estimated response timeslot length * 10 ms + + Safety timeout*. *Estimated response timeslot length* is the value based on expected length of data returned within the DPA response or it can be the worst case (e.g. 5 = 50 ms at STD mode). If the *Timeslot length* from the step 5 is equal to the diagnostic long timeslot (20 = 200 ms), then use the same value for the *Estimated response timeslot length*.
7. From the data length of the actual DPA response find out the *Actual response timeslot length*. Now the earliest time to send something to the IQMESH network is equal to: *Time the DPA confirmation was received + Hops * Timeslot length * 10 ms + Hops Response * Actual response timeslot length * 10 ms*. This time is used for waiting at the step 1.

Using this technique ensures reliable and optimal speed data delivery at the IQMESH network. Pay attention to the DPA requests that produce intentional delay at the addressed device side (e.g. UART Read/Write, SPI Read/Write, IO Set, OS Sleep, OS Reset).

### 2.5.3 DPA notification

DPA notification notifies a connected master device at the node side that there was a DPA request without a read-only (can be configured by bit EnableIFacenotificationOnRead) side-effect processed by the node. It consists of the same *foursome* that was part of the original DPA request except NAdr stores address of the sender, not addressee, and HwProfile contains actual HW Profile of device. DPA notification is therefore always 6 B long.

### 2.5.4 DPA response

DPA response is an actual answer to the DPA request. DPA response consists of the same foursome that was part of the original DPA request except the response message is flagged by the most significant bit of PCmd and HwProfile contains actual HW Profile of addressed device. Then come 2 bytes containing the Response code and DPA Value. In case of error (response code is NOT equal to `STATUS_NO_ERROR`) no additional data is present. In case of `STATUS_NO_ERROR` response code the presence of the additional data depends on the DPA response type.

When composing DPA response in the Custom DPA Handler there is sometimes a need to signalize an error response with certain Response Code. The way how to prepare such response is described at chapter Handle Peripheral Request.

### 2.5.5 Examples

**Note:**

DPA Value and data read from the memory shown in the following examples may be different in the real scenario.

**Example 1**

Switching on a red LED at coordinator:

- **DPA request** (master → slave)
  `NAdr=0x0000, PNum=0x06, PCmd=0x01, HwProfile=0xFFFF`
- **DPA response** (slave → master)
  `NAdr=0x0000, PNum=0x06, PCmd=0x81, HwProfile=0xABCD, Data={0x00}`(No error)`, {0x07}`(DPA Value)

***Notes:***

- NAdr 0x0000 Specifies coordinator address (0x00FC can be used too)
- PNum 0x06 Specifies red LED peripheral
- PCmd 0x01 Set LED On command
- DPA Value Coordinator's value

**Example 2**

Reading 2 bytes from RAM at address 1 of the local node:

- **DPA request** (master → slave)
  `NAdr=0x00FC, PNum=0x05, PCmd=0x00, HwProfile=0xFFFF, Data={0x01}`(Address)`, {0x02}`(Length)
- **DPA response** (slave → master)
  `NAdr=0x00FC, PNum=0x05, PCmd=0x80, HwProfile=0xABCD`
  `Data={0x00}`(No error)`, {0x07}`(DPA Value)`, {0xAB,0xCD}`(Read data)

***Notes:***

- NAdr 0x00FC Specifies local device address
- PNum 0x05 Specifies RAM peripheral
- PCmd 0x00 Read command
- DPA Value Local node's value

**Example 3**

Switching on a green LED at remote IQMESH node with address 0x0A:

- **DPA request** (master → slave)
  `NAdr=0x000A, PNum=0x07, PCmd=0x01, HwProfile=0xFFFF`
- **DPA confirmation** (slave → master)
  `NAdr=0x000A, PNum=0x07, PCmd=0x01, HwProfile=0xFFFF, Data={0xFF}`(Confirmation)`, {0x07}`(DPA Value)`,`
  `{0x06,0x03}`(Hops, Timeslot length)
- **DPA notification** (slave → master) at remote node side
  `NAdr=0x0000, PNum=0x07, PCmd=0x01, Data=<none>`
- **DPA response** (slave → master)
  `NAdr=0x000A, PNum=0x07, PCmd=0x81, HwProfile=0xABCD, Data={0x00}`(No error)`, {0x06}`(DPA Value)

***Notes:***

- PNum 0x07 Specifies green LED peripheral
- NAdr 0x0000 At DPA notification specifies that the Coordinator sent the original request
- DPA Value DPA confirmation: Coordinator's value
  DPA response: remote node's value

## 2.6 Device exploration

### 2.6.1 Peripheral enumeration

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0xFF | 0x3F | ? |

The HwProfile value is ignored at peripheral enumeration command.

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NAdr | 0xFF | 0xBF | ? | 0 | ? | DpaVer | | PerNr | StdPers | | | | HwProf | | HwProfV | | Flags |

DpaVer — DPA protocol version
- 1st byte: bits 0-6 = minor version, bit 7 = demo version
- 2nd byte: major version
  BCD coding is used, e.g. version 12.34 would be coded as 0x1234, i.e. 1st byte 0x34, 2nd byte 0x12

PerNr — Number of user defined peripherals

StdfPers — Bits array (starting from LSb of the 1st byte) specifying which of 32 standard peripherals were enabled in the HWP Configuration (it is a copy of first 4 bytes of the configuration area). If a peripheral is enabled in the configuration although it is not supported by the device, then calling Get peripheral information or Get information for more peripherals will return `PERIPHERAL_TYPE_DUMMY` peripheral type for this peripheral thus indicating that the peripheral is actually not available.
Bit values for IQMESH – Coordinator (bit 0) and IQMESH – Node (bit 1) peripherals are set according to the device support of these peripherals regardless of actual bit values stored at HWP Configuration.

HwProf — Hardware profile type, coded using little-endian style, 0x0000 if default

HwProfV — Hardware profile version, 1st byte = minor version, 2nd byte = major version

Flags
- bit.0 STD IQMESH RF Mode supported
- bit.1 LP IQMESH RF Mode supported
- bit.2 XLP IQMESH RF Mode supported
- bit.3-7 Reserved

**Example**

- **Request**
  `NAdr=0x0000, PNum=0xFF, PCmd=0x3F, HwProfile=0xFFFF`
- **Response**
  `NAdr=0x0000, PNum=0xFF, PCmd=0xBF, HwProfile=0xABCD, Data={0x00}`(No error)`, {0x07}`(DPA Value)`, {01,010}`(DpaVer 1.01)`, {01}`(PerNr)`, {E6,06,00,00}`(StdPers)`, {CD,AB}`(HwProf)`, {01,00}`(HwProfV)`, {41}`(Flags)

Coordinator (`NAdr=0x0000`) having 1 user defined peripheral, Hardware profile of type 0xABCD (version 0x0001), DPA version 0.1 (not a demo version) and these standard peripherals:
- 0x01 NODE
- 0x02 OS
- 0x05 RAM
- 0x06 LEDR
- 0x07 LEDG
- 0x09 IO
- 0x0A Thermometer
  bit array: `11100110.00000110.00000000.00000000`

### 2.6.2 Get peripheral information

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | PNum | 0x3F | ? |

The HwProfile value is ignored at peripheral information command.

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|
| NAdr | PNum | 0xBF | ? | 0 | ? | PerTE | PerT | Par1 | Par2 |

PerTE — Extended peripheral characteristic
PerT — Peripheral type. If the peripheral is not supported or enabled, then $PerT_x$ = `PERIPHERAL_TYPE_DUMMY`.
Par1 — Optional peripheral specific information
Par2 — Optional peripheral specific information

### 2.6.3 Get information for more peripherals

Returns the same information as Get peripheral information but for up to 14 peripherals of consecutive indexes starting with the specified PCmd.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0xFF | Per | ? |

*Per* — First peripheral from the list to get the information about

The HwProfile value is ignored at peripheral information command.

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 | 2 | 3 | … | 4*(n-1)+0 | 4*(n-1)+1 | 4*(n-1)+2 | 4*(n-1)+3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NAdr | 0xFF | RPer | ? | 0 | ? | $PerTE_1$ | $PerT_1$ | $Par1_1$ | $Par2_1$ | | $PerTE_n$ | $PerT_n$ | $Par1_n$ | $Par2_n$ |

RPer — Same as Per at request but with most significant bit set to indicate response message
n — Number of peripherals information was returned about.

If the peripheral at index x is not supported or enabled, then $PerT_x$ = `PERIPHERAL_TYPE_DUMMY`. The response data is always right-trimmed to the last supported or enabled peripheral that can fit in the data array i.e. the data never ends with one or more peripheral information with $PerT_x$ = `PERIPHERAL_TYPE_DUMMY`.

# 3 Peripherals

## 3.1 Standard operations

Commands marked *[sync]* are executed after IQMESH routing is finished thus this event is synchronized among all devices that handled the original DPA request. This applies to the DPA request being sent using broadcast address.

### 3.1.1 Writing to peripheral in general

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | | n - 1 |
|---|---|---|---|---|---|---|
| NAdr | PNum | PCmd | ? | $PData_0$ | … | $PData_{n-1}$ |

n Data length

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue |
|---|---|---|---|---|---|
| NAdr | PNum | PCmd | ? | 0 | ? |

PCmd Same as PCmd at request but with most significant bit set to indicate response message.

### 3.1.2 Reading from peripheral in general

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | PNum | PCmd | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | … | n - 1 |
|---|---|---|---|---|---|---|---|---|
| NAdr | PNum | PCmd | ? | 0 | ? | $PData_0$ | … | $PData_{n-1}$ |

PCmd Same as PCmd at request but with most significant bit set to indicate response message.
n Data length

## 3.2 IQMESH – Coordinator

PNum = 0x00

This peripheral is implemented at [C] and [NC] devices.

General note: bond state of the node is not synchronized between the node and coordinator. There are separated request for node and coordinator concerning the bonding.

### 3.2.1 Peripheral information

PerT `PERIPHERAL_TYPE_IQMESH_COORDINATOR`
PerTE `PERIPHERAL_TYPE_EXTENDED_READ_WRITE`
Par1 Maximum number of data (PData) bytes that can be sent in the DPA messages
Par2 0

### 3.2.2 Get addressing information

Returns basic network information.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x00 | 0x00 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 |
|---|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x80 | ? | 0 | ? | DevNr | Reserved |

DevNr Number of bonded network nodes

### 3.2.3 Get discovered nodes

Returns a bit map of discovered nodes.

Same as Get bonded nodes but `PCmd = 0x01`.

### 3.2.4 Get bonded nodes

Returns a bit map of bonded nodes.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x00 | 0x02 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | … | 31 |
|---|---|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x82 | ? | 0 | ? | $PData_0$ | … | $PData_{31}$ |

$PData_{0-31}$ Bit array indicating bonded nodes (addresses). Address 0 at $bit_0$ of $PData_0$, Address 1 at bit1 of $PData_0$ etc.

### 3.2.5 Clear all bonds

Removes all nodes from the list of bonded nodes at coordinator memory.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x00 | 0x03 | ? |

**Response**

Response: General response to writing request with `STATUS_NO_ERROR` Error code

### 3.2.6 Bond node

Bonds a new node by coordinator. There is a maximum approx. 10 s blocking delay when this function is called.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 |
|---|---|---|---|---|---|
| NAdr | 0x00 | 0x04 | ? | ReqAdr | Bonding mask |

ReqAdr A requested address for the bonded node. The address must not be used (bonded) yet. If this parameter equals to 0, then 1st free address is assigned to the node.
Bonding mask See IQRF OS User's and Reference guides (remote bonding, function `bondNewNodeRemote`).

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 |
|---|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x84 | ? | 0 | ? | BondAdr | DevNr |

BondAdr Address of the node newly bonded to the network
DevNr Number of bonded network nodes

### 3.2.7 Remove bonded node

Removes already bonded node from the list of bonded nodes at coordinator memory.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 |
|---|---|---|---|---|
| NAdr | 0x00 | 0x05 | ? | BondAdr |

BondAdr Address of the node to remove the bond to

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 |
|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x85 | ? | 0 | ? | DevNr |

DevNr Number of bonded network nodes

### 3.2.8 Re-bond node

Puts specified node back to the list of boded nodes in the coordinator memory.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 |
|---|---|---|---|---|
| NAdr | 0x00 | 0x06 | ? | BondAdr |

BondAdr Address of the node to be re-bonded

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 |
|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x86 | ? | 0 | ? | DevNr |

DevNr Number of bonded network nodes

### 3.2.9 Run discovery

Runs IQMESH discovery process. The time when the response is delivered depends highly on the number of network devices and the network topology thus it is not predictable. It can take from a few seconds to many minutes.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 |
|---|---|---|---|---|---|
| NAdr | 0x00 | 0x07 | ? | TxPower | MaxAddr |

TxPower TX Power used for discovery.
MaxAddr Specifies maximum node address to be part of the discovery process. This feature allows to split all node devices into tho parts: [1] devices having address from 0 to MaxAddr will be part of the discovery process thus they become routers, [2] devices having address from MaxAddr+1 to 239 will not be routers. See IQRF OS documentation for more information.
This parameter is ignored at demo version and value 5 is always used.

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 |
|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x87 | ? | 0 | ? | DiscNr |

DiscNr Number of discovered network nodes

### 3.2.10 Set DPA Param

Sets DPA Param. DPA Param (DPA Parameter) is one byte parameter stored at the coordinator RAM that configures network behavior. Default value 0x00 is set upon coordinator reset. Default value can be changed using Autoexec feature.

| Bit | Description | |
|---|---|---|
| 0-1 | Specifies which type of DPA Value is returned inside every DPA response or DPA confirmation messages: | |
| | *00* | *lastRSSI* IQRF OS variable * In case of {C} device the value is 0 until some RF packet is received. |
| | *01* | Value returned by `getSupplyVoltage()` IQRF OS call * |
| | *10* | Reserved * |
| | *11* | User specified DPA Value |
| 2 | If 1, it allows to easily diagnose the network behavior based on following LED activities. Please note that this activity might collide with LED peripheral when used simultaneously giving undesirable effects. | |
| | *Red LED flashes* | When Node or Coordinator receives network message |
| | *Green LED flashes* | When Coordinator sends network message or when Node routes network message |
| 3 | If 1, then instead of using ideal timeslot length a fixed 200 ms long timeslot is used. It allows easy tracking of network behavior. | |
| 4-7 | Reserved | |

* The highest 7th bit indicates, that the node, that returned the DPA response, provided a remote bonding to the another node. Then IQMESH - Node peripheral commands can be used to find out its module ID and proceed with node authorization using IQMESH – Coordinator peripheral.

DPA Param is transparently sent with every DPA message from the coordinator and thus it controls the network behavior “on the fly”. It is not permanently stored at nodes.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 |
|---|---|---|---|---|
| NAdr | 0x00 | 0x08 | ? | DPA Param |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 |
|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x88 | ? | 0 | ? | DPA Param |

DPA Param Previous value

### 3.2.11 Set Hops

Allows specifying fixed number of routing hops used to send the DPA request/response or to specify an optimization algorithm to compute number of routing hops. The default value 0xFF is set upon device reset.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 |
|---|---|---|---|---|---|
| NAdr | 0x00 | 0x09 | ? | Request Hops | Response Hops |

Hops values:
0x00, 0xFF: See a description of the parameter of function `optimizeHops()` in the IQRF documentation.
0x01 – 0xEF: Sets number of hops to the *Hops - 1*

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 |
|---|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x89 | ? | 0 | ? | Request Hops | Response Hops |

Hops Previous values

### 3.2.12 Discovery data

Allows to read coordinator internal discovery data. Discovery data can be used for instance for IQMESH network visualization and optimization. Discovery data structure is not public.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 |
|---|---|---|---|---|
| NAdr | 0x00 | 0x0A | ? | Address |

Address Address of the discovery data.

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | … | 15 |
|---|---|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x8A | ? | 0 | ? | Discovery data | | |

DiscoveryData Discovery data read from the coordinator private storage

### 3.2.13 Backup

Allows to read coordinator network info data that can be then restored to another coordinator in order to make a clone of the original coordinator. Backup data structure is not public.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 |
|---|---|---|---|---|
| NAdr | 0x00 | 0x0B | ? | Index |

Index Index of the block of data

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | … | 18 |
|---|---|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x8B | ? | 0 | ? | Network data | | |

Network data One block of the coordinator network info data

To read all data blocks just start with Index = 0 and execute Backup request. Then store received data block from the response. The 1st byte of the read data specifies how many data blocks remains to be read. So, if this byte is not 0 just increment Index (0, 1, …) and execute another Backup request.

### 3.2.14 Restore

Allows to write previously backed up coordinator network data to the same or another coordinator device. To execute the full restore all data blocks (in any order) obtained by Backup commands must be written to the device.

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

- Module IDs of the backed up coordinator and coordinator to restore to are identical.
- No network traffic comes from/to restored coordinator during restore process.
- Coordinator device is reset after whole restore is finished.
- It is recommended to run Run discovery command before 1st network use because of possible RF differences between new and provious coordinator device HW.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | … | 18 |
|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x0C | ? | Network data | | |

Network data One block of the coordinator network info data previously obtained by Backup command.

**Response:** General response to writing request with `STATUS_NO_ERROR` Error code

### 3.2.15 Authorize bond

Authorizes previously remotely bonded node. This give the node the final network address. See IQRF documentation for more information about remote bonding concept.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 | 2 |
|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x0D | ? | ReqAdr | Module ID | |

ReqAdr See Bond node request
Module ID Module ID (the lowest 2 bytes) of the node to be authorized. Module ID is obtained by calling Read remotely bonded module ID.

**Response:** see response of Bond node command (except PCmd is 0x8D).

### 3.2.16 Bridge

*[sync]* This command supported by [NC] devices allows to send and receive DPA requests and responses to and from the nested networks, respectively. The command must not be a part of the Batch, nor Autoexec. To bridge DPA request and response among more than one nested sub networks one bridge command can be nested inside another bridge command (see example below).

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 | 2 | 3 | 4 | 5 | 6 | ... | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x0E | ? | subNAdr | | subPNum | subPCmd | subHwProfile | | subPData | | |

subNAdr Network address of the device in the sub network controlled by the IQMESH – Coordinator of the [NC] device to send the DPA request to.
subPNum Peripehral number to send the DPA request to.
subPCmd DPA request command.
subHwProfile DPA request HW profile.
subPData Optional DPA request data depending on the actual subPCmd used.

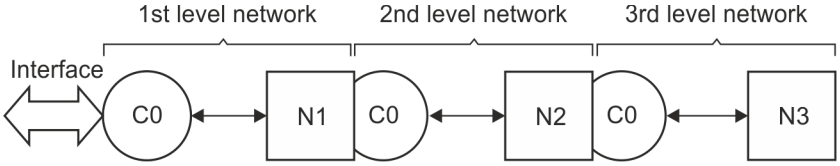
**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NAdr | 0x00 | 0x8E | ? | 0 | ? | subNAdr | | subPNum | subPCmd | subHwProfile | | subPData | | |

subNAdr, subPNum, subPCmd, subHwProfile, subPData are optional response data from each addressed node from every nested network. So for one original DPA request there is one response for each node i.e. when a node from 1st nested network is addressed then there is one response from the 1st node (NAdr) and another response from the node from the nested network (subPNum). There must be no other traffic in the participating networks in order to reliably deliver all DPA responses back to the main coordinator. Also note that every response being bridged from one network to the higher one has PData longer by 6 bytes. It must be ensured that the PData length at the very last response does not exceed the maximum allowed PData length.

**Example**

The following example sent from the main coordinator C0 pulses red LED of the node N3 at the 3rd nested network.




- **DPA request** (C0 → N1 → N2 → N3)
  NAdr=0x0001, PNum=0x00, PCmd=0x0E, HwProfile=0xFFFF, Data={0x0002}(N2 NAdr), {0x00}(Coordinator PNum), {0x0E}(Bridge Pcmd),{0xFFFF}(Bridge HwProfile), [{0x0003}(N3 NAdr), {0x06}(LEDR PNum), {0x03}(Pulse LED PCmd), {0xFFFF}(Pulse LED HwProfile)]
- **DPA response #1** (N1 → C0)
  NAdr=0x0001, PNum=0x00, PCmd=0x8E, HwProfile=0xFFFF, Data={0x00}(No error), {0x??}(DPA Value)
- **DPA response #2** (N2 → N1 → C0)
  NAdr=0x0001, PNum=0x00, PCmd=0x8E, HwProfile=0xFFFF, Data={0x00}(No error), {0x??}(DPA Value), {0x0002}(N2 Nadr), {0x00}(Coordinator PNum), {0x8E}(Bridge PCmd),{0x1234}(Bridge HwProfile)
- **DPA response #3** (N3 → N2 → N1 → C0)
  NAdr=0x0001, PNum=0x00, PCmd=0x8E, HwProfile=0xFFFF, Data={0x00}(No error), {0x??}(DPA Value), {0x0002}(N2 NAdr), {0x00}(Coordinator PNum), {0x8E}(Bridge Pcmd),{0x1234}(Bridge HwProfile), [{0x0003}(N3 NAdr), {0x06}(LEDR PNum), {0x83}(Pulse LED Pcmd),{0x5678}(Pulse LED HwProfile)]

### 3.2.17 Enable remote bonding

Implemented at [C] devices. Has the same behavior as Enable remote bonding except PCmd = 0x11.

### 3.2.18 Read remotely bonded module ID

Implemented at [C] devices. Has the same behavior as Read remotely bonded module ID except PCmd = 0x0F.

### 3.2.19 Clear remotely bonded module ID

Implemented at [C] devices. Has the same behavior as Clear remotely bonded module ID except PCmd = 0x10.

## 3.3 IQMESH – Node

PNum = 0x01

This peripheral is implemented at [N] and [NC] devices.

*General note:* Bond state of the node is not synchronized between the node and coordinator. There are separated requests for node and coordinator concerning the bonding.

### 3.3.1 Peripheral information

PerT `PERIPHERAL_TYPE_IQMESH_NODE`
PerTE `PERIPHERAL_TYPE_EXTENDED_READ_WRITE`
Par1 Maximum number of data (PData) bytes that can be sent in the DPA messages
Par2 0

### 3.3.2 Read

Returns IQMESH specific node information.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x01 | 0x00 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 - 10 | 11 |
|---|---|---|---|---|---|---|---|
| NAdr | 0x01 | 0x80 | ? | 0 | ? | ntwADDR … ntwCFG | Flags |

ntwADDR … ntwCFG Block of all ntw* IQRF OS variables (ntwADDR, ntwVRN, ntwZIN, ntwDID, ntwPVRN, ntwUSERADDRESS, ntwID, ntwVRNFNZ, ntwCFG) in the same order and size as located in the IQRF OS memory. See IQRF OS documentation for more information.
Flags Bit 0 indicates whether the Node device is bonded.

### 3.3.3 Remove bond

*[sync]* The bond is marked as unbonded (removed from network) using `removeBond()` IQRF call. Bonding state of the node at the coordinator side is not effected at all.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x01 | 0x01 | ? |

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.3.4 Enable remote bonding

Puts node into a mode, that provides a remote bonding of maximum one new node. Remote bonding gives the new node temporary network address (0xFE). A final logical network address is provided to the node using Authorize bond command. Then the node can be discovered thus giving its virtual routing number. See IQRF documentation for more information about remote bonding concept.

Node stays in the remote bonding node even if a new node was bonded. Then it allows only to the same node to be bonded again, bonding of other node is rejected. This gives possibility the new node to try bonding again in case when it did not receive bonding confirmation at previous bonding requests. Also see bit `ProvidesRemoteBonding`.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|
| NAdr | 0x01 | 0x04 | ? | Bonding mask | Control | User Data | |

Bonding mask See IQRF OS User's and Reference guides (remote bonding, function `bondNewNodeRemote`).
Control bit.0 enables remote bonding mode. If enabled then previously bonded node module ID is forgotten.
User Data Optional data that can be used at Reset Custom DPA Handler event.

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.3.5 Read remotely bonded module ID

This command returns module ID of the remotely bonded node. If no node was bonded then the command returns with `ERROR_FAIL`. If any node was bonded, then non-user DPA Values indicate it in every DPA response. See Set DPA Param. Also see bit RemoteBondingDone.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x01 | 0x02 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| NAdr | 0x01 | 0x82 | ? | 0 | ? | Module ID | | | | User Data | |

Module ID Module ID of the remotely bonded node. Bytes at position 0 and 1 can be used for bonding authorization later. See Authorize bond.
User Data Optional bonding user data specified at Reset Custom DPA Handler event.

### 3.3.6 Clear remotely bonded module ID

This call makes node to forget module ID of the node that was previously remotely bonded. After calling this command calling of Read remotely bonded module ID fails. This command does not affect remote bonding mode enable/disable state.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x01 | 0x03 | ? |

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.3.7 Remove bond address

*[sync]* The node stays in the IQMESH network (it is not unbonded) but a temporary address 0xFE is assigned to it. This allows to address it (them) or to authorize it later by AuthorizeBond. It is recommended to read the device's Module ID before removing bond address to be able to authorize it later.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x01 | 0x05 | ? |

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.3.8 Backup

Same as coordinator Backup except PCmd = 0x06.

### 3.3.9 Restore

Same as coordinator Restore except PCmd = 0x07.

## 3.4 OS

PNum = 0x02

### 3.4.1 Peripheral information

PerT `PERIPHERAL_TYPE_OS`
PerTE `PERIPHERAL_TYPE_EXTENDED_READ_WRITE`
Par1 Undocumented
Par2 Undocumented

### 3.4.2 Read

Returns some useful system information about the node.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x02 | 0x00 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 - 3 | 4 | 5 | 6-7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NAdr | 0x02 | 0x80 | ? | 0 | ? | ModuleID | OSVersion | TR&McuType | OsBuild | Rssi | SupplyVoltage | Flags |

ModuleID, OSVersion, TR&McuType, OsBuild — See `moduleInfo()` at IQRF OS Reference Guide
Rssi — See `lastRSSI` at IQRF Reference Guide. In case of {C} device the value is 0 until some RF packet is received.
SupplyVoltage — See `getSupplyVoltage()` at IQRF Reference Guide
Flags — Flags.0 is 1 if there is an insufficient OsBuild for the used DPA version.

### 3.4.3 Reset

*[sync]* Forces TR transceiver module to carry out reset.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x02 | 0x01 | ? |

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.4.4 Read HWP configuration

Reads a raw HWP configuration memory. Additional bytes being read are not documented.
Bit values for IQMESH – Coordinator (bit 0) and IQMESH – Node (bit 1) peripheral stored at HWP configuration are set the same way as at Peripheral enumeration.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x02 | 0x02 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | … | n |
|---|---|---|---|---|---|---|---|---|
| NAdr | 0x02 | 0x82 | ? | 0 | ? | Configuration | | |

Configuration Configuration read

### 3.4.5 Run RFPGM

*[sync]* Puts device into RFPGM mode with approx. 1 minute timeout. The device is reset when RFPGM process is finished or if it ends due to timeout. RFPGM runs at the same main channel (configured at HWP configuration) the network runs at.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x02 | 0x03 | ? |

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.4.6 Sleep

Puts device into sleep (power saving) mode.

*[sync]* This command is not implemented at the device having coordinator functionality i.e. [C] and [NC].

(In)accuracy of the real sleep time depends on the PIC LFINTOSC oscillator that runs watchdog timer. Oscillator frequency is mainly influenced by the device supply voltage and temperature volatility. See PIC MCU datasheet for more details.

If SPI interface is used then it is disabled before going to sleep and enabled after device wakes up.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 | 2 |
|---|---|---|---|---|---|---|
| NAdr | 0x02 | 0x04 | ? | Time | | Control |

Time Sleep time in 2.097s (i.e. 2048 * 1.024 ms) units. 0 specifies endless sleep (except *Control.bit1* is set to run calibration process without performing sleep). Maximum sleep time is 38 hours 10 minutes 38.95 seconds.

Control
- bit0 Wake up on PIN change. See IQRF sleep() method for more information.
- bit1 Runs calibration process before going to sleep. Calibration time takes approximately 132 ms and it is subtracted from the requested sleep time. Calibration time deviation may produce an absolute sleep time error at short sleep times. But it is worth to run the calibration always before a longer sleep because the calibration time deviation then accounts for a very small total relative error. The calibration is always run before a first sleep after the module reset if calibration was not already initiated by *Time=0* and *Control.bit1=1*.
- bit2 If set, then when the device wakes up after the sleep period, a green LED once shortly flashes. Useful for diagnostic purposes.

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.4.7 Batch

*[sync]* Batch command allows executing more individual DPA requests within one original DPA request. It is not allowed to embed Batch command itself within series of individual DPA requests. Using Run discovery is not allowed inside batch command list too.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | ... | n |
|---|---|---|---|---|---|---|
| NAdr | 0x02 | 0x05 | ? | DPA Requests | | 0 |

DPA Requests Contains typically more DPA requests to be executed. The format at which the DPA requests are stored is same as the format of Autoexec DPA requests. See Autoexec for more information.

**Example**

The following example runs simple broadcast set of DPA requests. It switches on red LED at devices with HW profile 0x1234 or green LED at devices with HW profile 0x5678 respectively, then waits for 200 ms (using I/O peripheral) and finally switches the same LEDs off.

```
NAdr=0x00FF, PNum=0x02, PCmd=0x05,HwProfile=0xFFFF, Data=
[1st command] {0x05(length), 0x06(PNum=LEDR), 0x01(PCmd=LED on), 0x1234(HwProfile)},
[2nd command] {0x05(length), 0x07(PNum=LEDG), 0x01(PCmd=LED on), 0x5678(HwProfile)},
[3rd command] {0x08(length), 0x09(PNum=I/O), 0x01(PCmd=Set),0xFFFF(HwProfile),0xFF(Delay
command),0x00C8(200ms)}
[4th command] {0x05(length), 0x06(PNum=LEDR), 0x00(PCmd=LED off),0x1234HwProfile)},
[5th command] {0x05(length), 0x07(PNum=LEDG), 0x00(PCmd=LED off),0x5678HwProfile)},
{0x00(end of batch)}
```

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.4.8 Set USEC/User Address

Sets value shared by both User Security Code (USEC) and User address. USEC is used for an additional authorization to enter maintenance DPA Service Mode. User address is used in case of 2 byte addressing (DFM2B), that is not supported yet by the DPA framework.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 |
|---|---|---|---|---|---|
| NAdr | 0x02 | 0x06 | ? | Value | |

Value A value to set USEC and User address. The initial value for a new device is 0xFFFF (65,535 decimal). Value is coded using little-endian style.

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.4.9 Set MID

Sets a unique Module ID (MID) of the device. This can be usefull for creating a backup HW of the coordinator device (also see coordinator Backup and Restore). A special encrypted 24 byte long key obtained from device manufacturer is needed. Nevetheless the very last 4 bytes equal to the current MID, and the previous 4 bytes equal to the new MID to be set.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 … 23 |
|---|---|---|---|---|
| NAdr | 0x02 | 0x07 | ? | Key |

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

## 3.5 EEPROM

PNum = 0x03

### 3.5.1 Peripheral information

PerT `PERIPHERAL_TYPE_EEPROM`
PerTE `PERIPHERAL_TYPE_EXTENDED_READ_WRITE`
Par1 Size in bytes. (In the current version of DPA equals to 160 at [N] device or 32 at [C] or [NC] devices.)
Par2 Maximum data block length. (In the current version of DPA equals to 32.)

Actual EEPROM address space starts at address 0x00 at [N] device or at 0x80 at [C] or [NC] devices.

### 3.5.2 Read

Reads data from memory.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 |
|---|---|---|---|---|---|
| NAdr | 0x03 | 0x00 | ? | Addr | Len |

Addr Address to read data from
Len Length of the data in bytes

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | … | n-1 |
|---|---|---|---|---|---|---|---|---|
| NAdr | 0x03 | 0x80 | ? | 0 | ? | $PData_0$ | … | $PData_{n-1}$ |

n Data length

### 3.5.3 Write

Writes data to memory.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 | … | n+1 |
|---|---|---|---|---|---|---|---|
| NAdr | 0x03 | 0x01 | ? | Addr | $PData_0$ | … | $PData_{n-1}$ |

PData Actual data to be written to the memory
Addr Address to write data to
n Data length

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

## 3.6 EEEPROM

PNum = 0x04

### 3.6.1 Peripheral information

| | |
|---|---|
| PerT | `PERIPHERAL_TYPE_EEPROM` |
| PerTE | `PERIPHERAL_TYPE_EXTENDED_READ_WRITE` |
| Par1 | Memory size in blocks (see Par2) (In the current version of DPA equals to 128 at [N] device or 16 at [C] or [NC] devices.) |
| Par2 | Data block size (equals to 16) |

Actual EEPROM address space starts at address 0x0000 at [N] device or at 0x0700 at [C] or [NC] devices.

### 3.6.2 Read & Write

See EEPROM with keeping these exceptions in mind:
- Addr unit is not byte but (zero based) block number
- Length unit is one byte and must be equal to the block size
- Actual available length of the EEEPROM peripheral differs between [N] and [C]+[NC] devices

## 3.7 RAM

PNum = 0x05

The address space of the periphehral occupies the whole bank 12 of the MCU RAM.

### 3.7.1 Peripheral information

| | |
|---|---|
| PerT | `PERIPHERAL_TYPE_RAM` |
| PerTE | `PERIPHERAL_TYPE_EXTENDED_READ_WRITE` |
| Par1 | Size in bytes. (In the current version of DPA equals to 48.) |
| Par2 | Maximum data block length. (In the current version of DPA equals to 48.) |

### 3.7.2 Read & Write

See EEPROM.

## 3.8 SPI (Slave)

PNum = 0x08

The peripheral is not available at the Coordinator [C] device.

### 3.8.1 Peripheral information

| | |
|---|---|
| PerT | `PERIPHERAL_TYPE_SPI` |
| PerTE | `PERIPHERAL_TYPE_EXTENDED_READ_WRITE` |
| Par1 | Maximum data block length |
| Par2 | Not used |

### 3.8.2 Write & Read

Writes and/or reads data to/from SPI interface. See UART Read & Write which uses the same read & write logic except PCmd = 0x00.

At LP and XLP modes the peripehral is enabled only during the time this command is executed. At STD mode the peripheral is enabled all the time except when device is at sleeping mode initiated by OS Sleep command.

## 3.9 LED

PNum = 0x06 or 0x07 for standard red respectively green LED.

### 3.9.1 Peripheral information

PerT `PERIPHERAL_TYPE_LED`
PerTE `PERIPHERAL_TYPE_EXTENDED_READ_WRITE`
Par1 `LED_COLOR_*` (* specifies one of the predefined color constants)
Par2 Not used

### 3.9.2 Set

Controls the state of the LED.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x06 or 0x07 | OnOff | ? |

OnOff 0x01 to switch LED on, 0x00 to switch LED off

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.9.3 Get

Returns a state of the LED.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x06 or 0x07 | 0x02 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 |
|---|---|---|---|---|---|---|
| NAdr | 0x06 or 0x07 | 0x82 | ? | 0 | ? | OnOff |

OnOff 0x01 when LED is on, 0x00 when LED is off

### 3.9.4 Pulse

Generates one LED pulse using IQRF OS function pulseLEDx().

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x06 or 0x07 | 3 | ? |

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

## 3.10 IO

PNum = 0x09

This peripheral controls IO pins of the MCU. Please note that the pins used by an internal IQRF TR module circuitry cannot be used and their control by this peripheral is blocked. See a corresponding IQRF TR module datasheet for the IO pins that are available.

### 3.10.1 Peripheral information

PerT `PERIPHERAL_TYPE_IO`
PerTE `PERIPHERAL_TYPE_EXTENDED_READ_WRITE`
Par1 Bit mask specifying supported MCU ports (b0=PORTA, b1=PORTB, ..., b7=PORTH)
Par2 Not used

### 3.10.2 Direction

This command sets the direction of the individual IO pins of the individual ports. Additionally the same command can be used to setup weak pull-ups at the pins where available. See datasheet of the PIC MCU for a description of IO ports.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 | 2 | … | n * 3 | n * 3 + 1 | n * 3 + 2 |
|---|---|---|---|---|---|---|---|---|---|---|
| NAdr | 0x09 | 0x00 | ? | $port_0$ | $mask_0$ | $value_0$ | | $port_n$ | $mask_n$ | $value_n$ |

port a) Specifies port to setup a direction to. 0x00=TRISA, 0x01=TRISB, …
or
b) Specifies port to setup a pull-up. 0x10=TRISA, 0x11=TRISB, …
mask Masks pins of the port.
value a) Actual direction bits for the masked pins. 0=output, 1=input.
or
b) Pull-up state. 0=disabled, 1=enabled.

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.10.3 Set

*[sync]* This command sets the output state of the IO pins. It also allows inserting an active waiting delay between IO pins settings. This feature can be used to generate an arbitrary time defined signals on the IO pins of the MCU. During the active waiting the device is blocked and any network traffic will not be processed.

This command is executed after the DPA response is sent back to the device that sent the original DPA IO Set request. Therefore if an invalid port is specified an error code is not returned inside DPA response but the rest of the request execution is skipped.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 | 2 | … | n * 3 | n * 3 + 1 | n * 3 + 2 |
|---|---|---|---|---|---|---|---|---|---|---|
| NAdr | 0x09 | 0x01 | ? | $command_0$ | | | | $command_n$ | | |

triple There are 2 types of 3 byte commands allowed:
1. Setting an output value
   port Specifies port to setup an output state. 0=PORTA, 1=PORTB, …
   mask Masks pins of the port to setup.
   value Actual output bit value for the masked pins.
2. Delay
   0xFF Specifies a delay command.
   delayL Lower byte of the 2 byte delay value, unit is 1 ms.
   delayH Higher byte of the 2 byte delay value, unit is 1 ms.

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.10.4 Get

This command is used to read the input state of all supported the MCU ports (PORTx).

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x09 | 0x02 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | … | N |
|---|---|---|---|---|---|---|---|---|
| NAdr | 0x09 | 0x82 | ? | 0 | ? | Port data | | |

Port data — Array of bytes representing state of port PORTA, PORTB, …, ending with the last supported MCU port.

**Example 1**

Setting of PORTA.0 and PORTC.2 as output, PORTC.3 as input.

- **Request**

  `NAdr=0x0001, PNum=0x09, PCmd=0x00, HwProfile=0xFFFF, Data={0x00`(PORTA)`, 0x01`(bit0=1)`, 0x00`(bit0=output)`} {0x02`(PORTC)`, 0x0C`(bit2=1, bit3=1)`, 0x08`(bit2=output, bit3=input)`}`

- **Response**

  `NAdr=0x0001, PNum=0x09, PCmd=0x80, HwProfile=0xABCD, Data={00}`(No error)`, {0x07}`(DPA Value)

**Example 2**

Setting of PORTA.0=1, PORTC.2=1, then wait for 300 ms, set PORTA.0=0.

- **Request**

  `NAdr=0x0001, PNum=0x09, PCmd=0x01, HwProfile=0xFFFF, Data={0x00`(PORTA)`, 0x01`(bit0=1)`, 0x01`(bit0=1)`} {0x02`(PORTC)`, 0x04`(bit2=1)`, 0x04`(bit2=1)`} {0xFF`(delay)`, 0x2C`(low byte of 300)`, 0x01`(high byte of 300)`} {0x00`(PORTA)`, 0x01`(bit0=1)`, 0x00`(bit0=0)`}`

- **Response**

  `NAdr=0x0001, PNum=0x09, PCmd=0x81, HwProfile=0xABCD, Data={00}`(No error)`, {0x07}`(DPA Value)

## 3.11 Thermometer

PNum = 0x0A for standard on-board thermometer peripheral

### 3.11.1 Peripheral information

PerT `PERIPHERAL_TYPE_THERMOMETER`
PerTE `PERIPHERAL_TYPE_READ`
Par1 Not used
Par2 Not used

### 3.11.2 Read

Reads on-board thermometer sensor value.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x0A | 0x00 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 | 2 |
|---|---|---|---|---|---|---|---|---|
| NAdr | 0x0A | 0x80 | ? | 0 | ? | TempC | Temp16 | |

TempC Temperature in °C, integer part, not rounded.
See return value of getTemperature() OS function. If the temperature sensor is not installed (see HWP Configuration) then the returned value is 0x80 = -128 °C.

Temp16 Complete 12 bit value of the temperature in 0.0625 °C units.
See `getTemperature()` OS function. If the temperature sensor is not installed the value is undefined.

## 3.12 PWM

PNum = 0x0B for standard MCU PWM peripheral

The peripheral is available at Demo version only and only at the [N] device.

### 3.12.1 Peripheral information

PerT `PERIPHERAL_TYPE_PWM`
PerTE `PERIPHERAL_TYPE_WRITE`
Par1 Not used
Par2 Not used

### 3.12.2 Set

Sets PWM parameters.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 | 2 |
|---|---|---|---|---|---|---|
| NAdr | 0x0B | 0x00 | ? | Prescaler | Period | Duty |

Prescaler
- Bits<1:0> codes four values for CCP6CON register:
  - 11 = prescaler is 64
  - 10 = prescaler is 16
  - 01 = prescaler is 4
  - 00 = prescaler is 1
- Bits<5:4> codes two least significant bits of 10bit Duty cycle <1:0>

Period Sets the PR6 register for PWM period

Duty Eight most significant bits of 10bit duty cycle value <9:2>. It sets the register CPR6

When all 3 parameters equal to 0, PWM is stopped.

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

**Example 1**

Set PWM for 1 kHz with 50% of duty cycle and prescaler 16:

- **DPA request** (master > slave)
  `NAdr=0x0000, PNum=0x0B, PCmd=0x00, HwProfile=0xFFFF, Data={0x02,0x7d,0x40}`
- **DPA response** (slave > master)
  `NAdr=0x0000, PNum=0x0B, PCmd=0x80, HwProfile=0xABCD, Data={0x00}` (No error)

**Example 2**

Set PWM for 1 kHz with 70% of duty cycle and prescaler 16.

Note: prescaler value is 0x02 = 0b00000010, but the duty cycle value is in this case 0x15E = 0b101011110, the bites<1:0> (0b101011**10**) are added into Prescaler value (0b00**10**0010 = 0x22) to bits <5:4> and the seven most significant bits (0b**1010111**10) are written into Duty (0b1010111 = 0x57).

- **DPA request** (master > slave)
  `NAdr=0x0000, PNum=0x0B, PCmd=0x00, HwProfile=0xFFFF, Data={0x22,0x7d,0x57}`
- **DPA response** (slave > master)
  `NAdr=0x0000, PNum=0x0B, PCmd=0x80, HwProfile=0xABCD, Data={0x00}` (No error)

## 3.13 UART

PNum = 0x0C for standard UART peripheral

The peripheral is not available at the Coordinator [C] device and is not supported at LP and XLP modes.

### 3.13.1 Peripheral information

PerT `PERIPHERAL_TYPE_UART`
PerTE `PERIPHERAL_TYPE_READ_WRITE`
Par1 Maximum data block length
Par2 Not used

### 3.13.2 Open

This command opens UART at specified baudrate and flushes internal read and write buffers. The size of the read and write buffers is 32 bytes.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 |
|---|---|---|---|---|
| NAdr | 0x0C | 0x00 | ? | BaudRate |

BaudRate specifies baud rate:

- 0x00 1200 baud
- 0x01 2400 baud
- 0x02 4800 baud
- 0x03 9600 baud
- 0x04 19200 baud
- 0x05 38400 baud
- 0x06 57600 baud
- 0x07 115200 baud
- other returns `ERROR_DATA`

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

**Example 1**

Open UART for communication with 9600 baud rate:

- **DPA request** (**master** > slave)
  `NAdr=0x0000`, `PNum=0x0C`, `PCmd=0x00`, `HwProfile=0xFFFF`, `Data={0x02}`(9600 baud)
- **DPA response** (slave > master)
  `NAdr=0x0000`, `PNum=0x0C`, `PCmd=0x80`, `HwProfile=0xABCD`, `Data={0x00}`(No error)

### 3.13.3 Close

Closes UART interface.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x0C | 0x01 | ? |

**Response**

General response to writing request with `STATUS_NO_ERROR` Error code.

### 3.13.4 Write & Read

Reads and/or writes data to/from UART interface. If UART is not open, the request fails with ERROR_FAIL.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 | … | n |
|---|---|---|---|---|---|---|---|
| NAdr | 0x0C | 0x02 | ? | ReadTimeout | WrittenData | | |

ReadTimeout Specifies timeout in 10 ms unit to wait for data to be read from UART after data is (optionally) written. 0xff specifies that no data should be read.
WrittenData Optional data to be written to the UART
n Number of bytes to be written.

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | … | n-1 |
|---|---|---|---|---|---|---|---|---|
| NAdr | 0x0C | 0x82 | ? | 0 | ? | ReadData | | |

ReadData Optional data read from UART if the reading was requested and data is available.
n Number of bytes that was read.
Please note that internal buffer limits maximum number of bytes to PERIPHERAL_UART_MAX_DATA_LENGTH.

**Example 1**

Write three bytes (0x00, 0x01, 0x02) to UART, no reading:

- **DPA request** (master > slave)
  `NAdr=0x0000, PNum=0x0C, PCmd=0x02, HwProfile=0xFFFF, Data={0xff}`(No reading) `{0x00,0x01,0x02}`
- **DPA response** (slave > master)
  `NAdr=0x0000, PNum=0x0C, PCmd=0x82, HwProfile=0xABCD, Data={0x00}`(No error)

**Example 2**

Write three bytes (0x00, 0x01, 0x02) to UART, read 4 bytes after 10 ms:

- **DPA request** (master > slave)
  `NAdr=0x0000, PNum=0x0C, PCmd=0x02, HwProfile=0xFFFF, Data={0x01}`(10ms timeout)
  `{0x00,0x01,0x02}`(written data)
- **DPA response** (slave > master)
  `NAdr=0x0000, PNum=0x0C, PCmd=0x82, HwProfile=0xABCD,`
  `Data={0x00}`(No error) `{0xaa,0xbb,x0cc,0xdd}`(read data)

## 3.14 FRC

PNum = 0x0D for standard FRC peripheral.

The peripheral is available at the [C] and [NC] devices.

### 3.14.1 Peripheral information

PerT `PERIPHERAL_TYPE_FRC`
PerTE `PERIPHERAL_TYPE_READ_WRITE`
Par1 Not used
Par2 Not used

### 3.14.2 Send

This command starts Fast Response Command (FRC) process supported by IQRF OS. It allows quickly and using only one command to collect same type of information from multiple nodes in the network. Type of the collected information is specified by a byte called FRC command. Currently IQRF OS allows to collect either 2 bits from all (up to 239) nodes or 1 byte from up to 62 nodes having logical addresses 1-62. Former is selected when FRC command has its bit.7 = 0, the latter by bit.7 = 1. Bits 0-6 of the FRC command byte actually specifies the type of the information. When bits are collected, then the 1st bits from the nodes are stored in the bytes of index 0-29 of the output buffer, 2nd bits from the nodes are stored in the bytes of index 32-61. When bytes are collected then they are stored at bytes 1-62 of the output buffer. For more information see IQRF OS manuals. If node does not for some reason return a FRC value at all, then either returned bits or bytes are equal to 0.

**Request**

| NAdr | PNum | PCmd | HwProfile | 0 | 1 .. 2 |
|---|---|---|---|---|---|
| NAdr | 0x0D | 0x00 | ? | FRC Command | UserData |

FRC Command Specifies data to be collected.
UserData Used data that are available at IQRF OS variable `DataOutBeforeResponseFRC` at FRC Value event.

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | 1 | … | N |
|---|---|---|---|---|---|---|---|---|---|
| NAdr | 0x0D | 0x80 | ? | 0 | ? | Status | FRC data | | |

Status Return code of the sendFRC() IQRF OS function. See IQRF OS documentation for more information.
FRC data Data collected from the nodes. Because the current version of DPA cannot transfer the whole FRC output buffer at once (currently only up to 55 bytes), the remaining bytes of the buffer can be read by the next described command.

### 3.14.3 Extra result

Reads remaining bytes of the FRC result, so the total number of bytes obtained by both commands will be total 64. It is recommended to call this command immediately after the FRC Send command to preserve previously collected FRC data.

**Request**

| NAdr | PNum | PCmd | HwProfile |
|---|---|---|---|
| NAdr | 0x0D | 0x01 | ? |

**Response**

| NAdr | PNum | PCmd | HwProfile | ErrN | DpaValue | 0 | … | N |
|---|---|---|---|---|---|---|---|---|
| NAdr | 0x0D | 0x81 | ? | 0 | ? | FRC data | | |

FRC data Remaining FRC data that could not be read by FRC Send command because DPA data buffer size limitations.

### 3.14.4 Predefined FRC Commands

- **FRC_Prebonding = 0x00**

Collects bits. Bit 0 is 1 when node is accessible, bit1 is 1 if the node provided pre-bonding to a new node. If bit 0 of user data is set, the remote bonding at node device is also disabled. Subsequently detail information can be read using Read remotely bonded module ID from the node.

- **FRC_UART_SPI_data = 0x01**

Collects bits. Bit 0 is 1 when node is accessible, bit1 is 1 when there is some data available for reading from UART or SPI peripheral.

- **FRC_Temperature = 0x80**

Collects bytes. Result byte equals to the temperature value read by *getTemperature()* IQRF OS method. If resulting temperature is 0°C, that would normally equal to value 0, then a fixed value 0x7F is returned instead. This makes possible to distinguish between devices reporting 0°C and devices not reporting at all. Device would normally never return a temperature corresponding to the value 0x7F, because +127°C is out of working temperature range.

# 4 HWP Configuration

HWP (hardware profile) configuration is stored at Flash memory of the MCU. The configuration can be modified only by IQRF IDE using SPI or RFPGM programming or using DPA Service Mode. It is necessary to correctly configure the device before DPA is used for the first time.

| Address | Description |
|---|---|
| 01 | Array of 32 bits. Each bit enables/disables one of the standard 32 predefined peripherals. Peripheral #0 (Coordinator) is controlled by bit 0.0, peripheral #31 (currently not used, but reserved) is controlled by bit 3.7. It does not make sense to enable the peripheral that is not implemented in the currently used device (see Peripheral enumeration). |
| 02 | |
| 03 | |
| 04 | |
| 05 | Various DPA configuration flag bits: |
| bit 0 | If set, then a Custom DPA handler is called in case of an event. The handler can define user peripherals, handle messages to standard peripherals and add special used defined device behavior. |
| bit 1 | If set, then Node device can be controlled by a local interface. In this case the same peripheral must not be enabled. This option is not valid for a main network coordinator device [C] and is not supported at LP and XLP modes. |
| bit 2 | If set, then DPA Autoexec is run at the module boot time. |
| bit 3 | If set, then the Node device does not route packet on the background. |
| 06 | RF channel A of the optional subordinate network in case the node also plays a role of the coordinator of such network. Such network can be controlled by [NC] device. Valid numbers depend on used RF band. |
| 07 | Same as above but second B channel. |
| 08 | RF output power. Valid numbers 0-7. |
| 09 | RF signal filter. Valid numbers 0-64. |
| 0A | Timeout when receiving RF packets at LP or XLP modes. Unit is cycles (one cycle is 46 ms at LP, 770 ms at XLP mode). Greater values save energy but might decrease responsiveness to the master interface DPA Requests and also decrease Idle event calling frequency. Valid numbers 1-255. Also see API variable uns8 LP_XLP_toutRF. |
| 0B | Baud rate of the UART interface if used. Uses the same coding as UART Open (i.e. 0x00 = 1200) |
| 11 | RF channel A of the main network. Valid numbers depend on used RF band. |
| 12 | Same as above but second B channel. |

# 5 Autoexec

When Autoexec is enabled, then a series of DPA requests can be executed at the boot time (after reset) of the device. DPA requests are stored at the block at the external EEPROM starting from its physical address 0x7c0 (the array is located at the very end of the external EEPROM address space as well as at the very end of the EEEPROM DPA Peripheral; size of the block is 64 bytes). When addressing this EEPROM space by DPA EEEPROM peripheral please note that the actual address used will differ between node or coordinator devices as the amount of coordinator available external EEPROM space is limited for the EEEPROM peripheral. DPA requests are stored next to each other and are structured according DPA protocol. There is one exception - a total size of the DPA request in bytes is stored at the place of a corresponding NAdr (in this case it is only 1 byte wide, not 2 bytes as normal NAdr). 0x00 is stored after the very last DPA request to indicate the end of Autoexec batch. When executing DPA request a local interface notification is not performed although DPA via interface is enabled. Other events at the user DPA routine are called as usual. It is not allowed to embed Batch within series of individual DPA requests.

**Example**

The following example shows the bytes stored at the Autoexec external EEPROM memory that will run these 4 actions upon the module reset:

1. Switch the green LED On (PNum=0x07)
2. Open UART at 9600 baud rate (PNum=0x0C)
3. Write hex. bytes [01,02,03,04,05] to the UART (PNum=0x0C)
4. Write hex. bytes [06,07,08,09,0a] to the RAM at address 0x0A (PNum=0x05)

Actual bytes stored at serial EEPROM from address 0x7c0:

```
   Len   PNum PCmd     HwProfile     Data
1. 0x05, 0x07, 0x01(On)  ,0xFFFF
2. 0x06, 0x0C, 0x00(open) ,0xFFFF, 0x03(9600 baud)
3. 0x0b, 0x0C, 0x02(write) ,0xFFFF, 0xff(no UART read), {0x01, 0x02, 0x03, 0x04, 0x05}(data)
4. 0x0b, 0x05, 0x01(write) ,0xFFFF, 0x0a(address), {0x06, 0x07, 0x08, 0x09, 0x0a}(data)
5. 0x00(end of Autoexec)
```

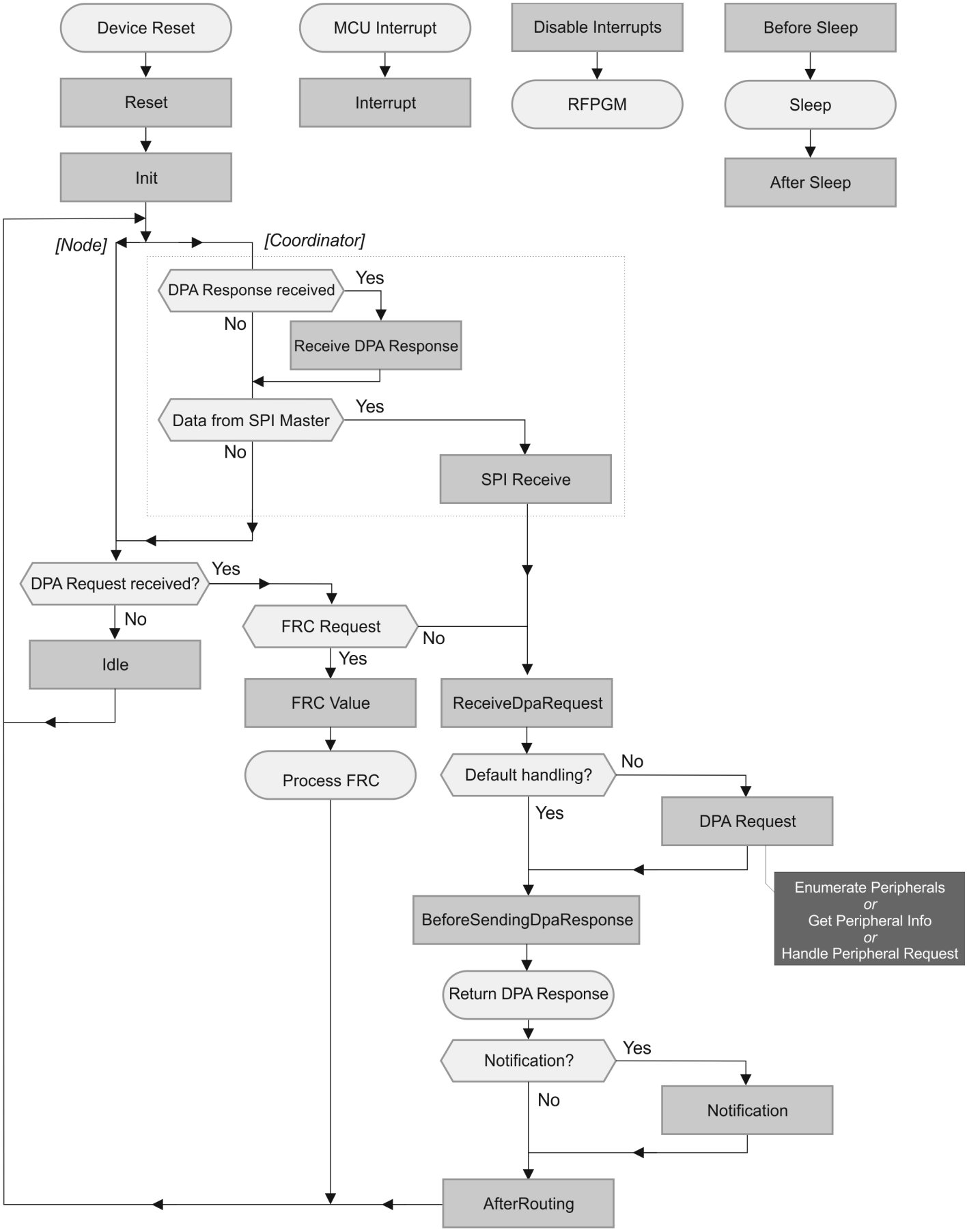
# 6 Custom DPA Handler

Custom DPA handler is an optional C routine that can handle various events and thus implements user peripherals, handles standard peripherals, provides peripheral virtualization, adds internal device logic and much more. If the custom DPA handler is implemented it must be enabled in the HWP configuration in order to receive events. Symbols, variables, structures, methods etc. needed to implement custom DPA handler are defined at header files `DPA.h` and `DPAcustomHandler.h`.

Please respect the following rules when implementing Custom DPA handler:

1. Custom DPA handler must be the first C routine implemented in your code.
2. There is a 736 instruction long block in the MCU flash memory to implement custom DPA handler in the current version od DPA.
3. “Cases” for unhandled events do not have to be programmed to save memory space and make code more readable.
4. Variables as well as function parameters must be allocated in the standard RAM bank 11 only. The whole bank is available.
5. Do not use `bufferRF`, `bufferCOM`, `bufferINFO` and `bufferAUX` at all (except inside events `Reset`, `Init`, `Idle` and `DisableInterrupts`).
6. Also do not use `userReg0` and `userReg1` variables unless you do not call any DPA API function.
7. Maintain the written code as much speed optimized as possible as the long time spent in the user code might negatively influence device behavior. Especially Interrupt and Idle events must be programmed extremely effectively.
8. Special attention must be paid to the implementation of an Interrupt event. See details in the dedicated chapter.
9. Do not use timer TMR6 at the coordinator only device [C].
10. Do not use IQRF methods `start[Long]Delay()` and `waitDelay()` (except events `Reset`, `Init`, `Idle` and `DisableInterrupts`). Use waitMS() instead.
11. Sending and receiveing packets is allowed only at events `Reset`, `Init`, `Idle`, `DisableInterrupts` and `AfterRouting`. It is required to keep same RF settings (see *setTXpower, setRFspeed, setRFband, setRFchannel, setRFmode, set*mode, setNetworkFiltering*, setRouting*,* etc. IQRF OS functions) that were set at the beginning of the event upon the event exit.
12. Do not modify content of IQRF OS variables within event code. It is required to save their values and restore them at the event exit.

The following flow chart depicts main events flow:



Next paragraphs describe available events in more detail. If not other specified then the return value from the routine does not matter. The code fragments are for the illustration purpose only. Please use the C code template distributed with DPA package instead.

Typical skeleton of the Custom DPA Handler looks like this:

```
// Default IQRF include
#include "../includes/template-basic.h"

// Uncomment to implement Custom DPA Handler for Coordinator
//#define COORDINATOR_CUSTOM_HANDLER

// Default DPA header
#include "DPA.h"
// Default Custom DPA Handler header
#include "DPAcustomHandler.h"

// Real Custom DPA Handler function
bit CustomDpaHandler ()
{
  // Detect DPA event to handle
  switch ( GetDpaEvent() )
  {
  case DpaEvent_Interrupt:
       // ...
       return TRUE;

// Other events ...

  case DpaEvent_Idle:
       // ...
       return FALSE;

  case DpaEvent_None:
      if ( IsDpaEnumPeripheralsRequest() )
       // Enumerate Peripherals
      {
        // ...
        return TRUE;
      }
      else if ( IsDpaPeripheralInfoRequest() )
      //  Get Peripheral Info
      {
        // ...
        return TRUE;
      }
      else
      //  Peripheral Request
      {
        // ...
        return TRUE;
      }
  }
}

// Default Custom DPA Handler header
// (2nd include to implement Code bumper to detect too long code of the Custom DPA
Handler)
#include "DPAcustomHandler.h"
```

## 6.1 Interrupt

This event is not raised in demo version. The event is called whenever an MCU interrupt occurs. It is not implemented at the coordinator only device [C].

Please make sure the following rules are met when implementing this event:

- The time spent handling this event is critical. If there is no interrupt to handle return immediately otherwise keep the code as fast as possible.
- Only global variables or local ones marked by "static" keyword can be used to allow reentrancy.
- Make sure race condition does not occur when accessing those variables at other places.
- Make sure (inspect .lst file) compiler did not create any hidden temporary local variable (occurs when using division, multiplication or bit shifts).
- Do not call any OS functions except `getINFx()` and `setINDFx()`.
- Do not use any OS variables.
- All above rules apply also to any other function being called, although calling any function from interrupt event is not recommended.

**Example**

```
case DpaEvent_Interrupt:
     if (!TMR6IF)
       return TRUE;
     TMR6IF = 0;
     T6CON = 0b0.0110.1.00;
     timerOccured = TRUE; // timerOccured is a global variable
     return TRUE;
```

## 6.2 Idle

This event is periodically raised when a main loop is waiting for incoming RF (or interface) message to handle. The event is also used to obtain user DPA Value that is transferred by `userReg0` variable. The time spent handling this event is critical. When there is not traffic then the event is called approximately every 420 µs in STD mode.

Note that the frequency at which the event is called depends mainly on the time spend inside RFRXpacket() IQRF OS function located in the main DPA loop. The worst case is when there is full IQMESH network consisting of 239 devices and the long diagnostic timeslot (200 ms) is used. In this case the Idle events might not be called even once in 239 × 200 ms = 47.8 s.

If RF channel and mode are changed by a user code they do not have to be restored back at [NC] devices as they are regularly updated inside the main application loop.

**Example**

```
case DpaEvent_Idle:
     // Go sleep?
     if ( sleepTime != 0 )
     {
          // Prepare OS Sleep DPA Request
          // Time in 2.097s units
          _DpaMessage.PerOSSleep.Time = sleepTime;
          sleepTime = 0;
          _PNum = PNUM_OS;
          _PCmd = CMD_OS_SLEEP;
          // LEDG flash after wake up
          _DpaMessage.PerOSSleep.Control = 0b100;
          _DpaDataLength = sizeof( _DpaMessage.PerOSSleep );
          // Perform local DPA Request
          // BeforeSleep and AfterSleep events will not be called in this case!
          DpaApiLocalRequest();
     }
     // Return DPA value
     userReg0 = myUserDpaValue;
     return FALSE;
```

## 6.3 Init

This event is called just before the main loop starts. Interrupt is enabled so the Interrupt event can be already called. Also Enumerate Peripherals is called before this event is raised in order to find out the hardware profile. Immediately after the event is processed the Autoexec is executed. This event is typically used to initialize peripherals and global variables.

If variable bit NodeWasBonded is set, then variable `DataOutBondRequestAdvanced` contains user data passed from node that provided pre-bonding of the device.

**Example**

```
case DpaEvent_Init:
     myVariable = 123;
     T6CON = 0b0.0110.1.00;
     TMR6IE = 1;
     return FALSE;
```

## 6.4 Notification

This event is called when a DPA request was successfully processed and the DPA response was sent. DPA response (but not original request) is available at this event . User can sense what peripheral was accessed and react accordingly.

**Example**

```
case DpaEvent_Notification:
     // Anything was writen to the RAM?
     if (_PNum == PNUM_RAM && _PCmd == CMD_RAM_WRITE)
     {
       if (PeripheralRam[0] == 0xAB)
           LEDR = 1;
       else
           LEDG = 1;
       ramWritten = TRUE;
     }
     return FALSE;
```

## 6.5 AfterRouting

*[sync]* This event is called after the DPA response was sent and (optional) Notification event and (optional) Interface Notification is sent. In any case the packet routing of the original DPA request is finished.
Please note that the RF channel is not defined but if it is changed by a user code (e.g. before calling DpaApiRfTxDpaPacket) its value must be restored. Also note that the original DPA request nor response foursome as well as DPA data are not available any more.

Example

```
case DpaEvent_AfterRouting:
     if ( ramWritten )
     {
       ramWritten = FALSE;
       LEDR = 0;
       LEDG = 0;
     }
     return FALSE;
```

## 6.6 BeforeSleep

This event is called before device goes to the Sleep mode.

This event is not implemented at the device having coordinator functionality i.e. [C] and [NC] and not in demo version.

**Example**

```
case DpaEvent_BeforeSleep:
     StopMyPeripherals();
     return FALSE;
```

## 6.7 AfterSleep

This event is called after device wakes up from the Sleep mode.

This event is not implemented at the device having coordinator functionality i.e. [C] and [NC] not in demo version.

**Example**

```
case DpaEvent_AfterSleep:
     StartMyPeripherals();
     return FALSE;
```

## 6.8 Reset

This event is not raised in demo version. The event is called just after the module was reset. It can be used to handle bonding/unbonding of the node in [N] and [NC] devices. In this case the code must return TRUE. If node is not bonded the handler routine must not finish until the node is bonded.

The event is also used to specify optional Bonding user data (see code example below) using variables `DataInBondRequestAdvanced` and `DataOutBondRequestAdvanced` in [N] and [NC] devices that is passed during remote bonding process and can be read by Read remotely bonded module ID. The code should also handle setting of bit NodeWasBonded.

**Example**

```
case DpaEvent_Reset:
     if (!doCustomBonding)
     {
          DataInBondRequestAdvanced = 0xABCD;
          return FALSE;
     }

     if (amIBonded())
     {
        if (unBondCondition)
        {
           removeBond();
           _LEDR = 1;
           waitDelay(100);
           _LEDR = 0;
        }
     }
     else
     {
       while (!amIBonded())
       {
          if (bondRequestCondition)
          {
            DataInBondRequestAdvanced = 0x1234;
            bondRequestAdvanced();
            setWDToff();
          }
       }

       NodeWasBonded = TRUE;
       bondingUserDataOut = DataOutBondRequestAdvanced;
     }

    return TRUE;
```

## 6.9 Disable Interrupts

This event is not raised in demo version. The event is called when device needs all hardware interrupts to be disabled. Such moment occurs e.g. just before entering RFPGM mode by Run RFPGM command.

**Example**

```
case DpaEvent_DisableInterrupts:
    // ADC Interrupt Enable - off
    ADIE = 0;
    return FALSE;
```

## 6.10 FrcValue

*[sync]* This event is called whenever the node is asked to provide data to be collected by FRC (see Send) and specified FRC Command is not handled by DPA itself (see Predefined FRC Commands). FRC Command value is accessible at MPRW1 IQRF OS variable. FRC data to be collected must be stored at responseFRCvalue IQRF OS variable. If bits are collected then only lowest 2 bits are used. Before calling the event the variable is prefilled with value 0x01. It is important that the code will take the same time at all nodes in order to keep their synchronization (event is fired at the same time at all nodes). User data passed by Send are accessible at `DataOutBeforeResponseFRC` IQRF OS variable.

This event is implemented at [N] and [NC] devices.

**Example**

```
case DpaEvent_FrcValue:
  {
      // This example is sensitive to the bit FRCommand 0x33
      if ( MPRW1 == 0x33 )
      {
        // Return info about providing remote bonding
        if ( ProvidesRemoteBonding )
            // Both bits bit0 and bit1 are set now
            responseFRCvalue.1 = 1;
      }
      // This example is sensitive to the byte FRCommand 0x83
      else if ( MPRW1 == 0x83 )
      {
        // Just return your logical address as an example
        responseFRCvalue = ntwADDR;
      }

      return FALSE;
  }
```

## 6.11 ReceiveDpaResponse

This event is called when there is a DPA response packet received from the network. If the event handler returns TRUE, then further standard DPA response processing (passing DPA response to the interface master internally by DpaApiSendToSpiMaster) is skipped. The event is raised even when HwProfile does not match.

This event is implemented at [C] and [NC] devices but not in demo version.

**Example**

```
case DpaEvent_ReceiveDpaResponse:
  {
      // This example just for demonstration purposes consumes any
      // DPA response CMD_LED_PULSE at peripheral PNUM_LEDG and pulses LEDR locally
      if ( _PNum == PNUM_LEDG && _PCmd == ( CMD_LED_PULSE | RESPONSE_FLAG ) )
      {
        pulseLEDR();
        return TRUE;
      }

      return FALSE;
  }
```

## 6.12 IFaceReceive

This event is called when there is a DPA request packet received from the interface master. If the event handler returns TRUE, then further standard DPA request processing (sending DPA confirmation back to the interface master, passing DPA response to the network internally by DpaApiRfTxDpaPacketCoordinator) is skipped. In this case interface master receives an error DPA response with `ERROR_INTERFACE_CUSTOM_HANDLER` Response Code. The event is raised even when HwProfile does not match.

This event is implemented at [C] device but not in demo version.

**Example**

```
case DpaEvent_IFaceReceive:
  {
      // This example just for demonstration purposes consumes any DPA Request
      // CMD_LED_PULSE at peripheral PNUM_LEDR and pulses LEDG locally
        if ( _PNum == PNUM_LEDR && _PCmd == CMD_LED_PULSE )
        {
            pulseLEDG();
            return TRUE;
        }

        return FALSE;
  }
```

## 6.13 ReceiveDpaRequest

This event is not raised in demo version. The event is called when a DPA request (except Get information for more peripherals) is received from network or from interface master (if applicable). If the event handler returns TRUE, then the request is not passed to the default handling by DPA Request event. In this case the programmer is fully responsible for preparing a valid DPA Response that will be returned to the device that sent original DPA request. The event is raised even when HwProfile does not match.

**Example**

```
case DpaEvent_ReceiveDpaRequest:
  // Returns error when there is an attempt to write to the address 0 of RAM peripheral
  if ( _PNum == PNUM_RAM && _PCmd == CMD_RAM_WRITE && _DpaMessage.MemoryRequest.Addr == 0 )
  {
    _DpaMessage.ErrorAnswer.ErrN = ERROR_FAIL;
    _DpaMessage.ErrorAnswer.PNumOriginal = PNUM_RAM;
    _PNum = PNUM_ERROR_FLAG;
    _PCmd = CMD_RAM_WRITE | RESPONSE_FLAG;
    _DpaDataLength = sizeof( _DpaMessage.ErrorAnswer );
    return TRUE;
  }
  return FALSE;
```

## 6.14 BeforeSendingDpaResponse

This event is not raised in demo version. The event is called when a DPA response (except response to Get information for more peripherals) is ready to be returned to the device that sent a DPA request via network or from the interface master (if applicable). The event handler can inspect or modify the DPA response event in the way that the error code is returned.

**Example**

```
case DpaEvent_BeforeSendingDpaResponse:
  // Always adds one more read byte from EEEPROM peripheral and sets it to 0x55
  if ( _PNum == PNUM_EEEPROM && _PCmd == CMD_RAM_READ )
  {
      _DpaDataLength++;
      FSR0 = _DpaMessage.Response.PData + _DpaDataLength - 1;
      setINDF0( 0x55 );
  }
  return FALSE;
```

**Example**

```
case DpaEvent_BeforeSendingDpaResponse:
  // This example hides even enabled and implemented PNUM_IO peripheral
  if ( IsDpaEnumPeripheralsRequest() )
    _DpaMessage.EnumPeripheralsAnswer.StandardPer[ PNUM_IO / 8 ] &= ~( 1 << ( PNUM_IO % 8 ) );
  else
  if ( _PNum == PNUM_IO && _PCmd == CMD_GET_PER_INFO )
    _DpaMessage.PeripheralInfoAnswer.PerT = PERIPHERAL_TYPE_DUMMY;
  return FALSE;
```

## 6.15 DPA Request

DPA requests are handled in the same way as the build-in DPA interpreter does it. DPA request is passed when there is no special event signaled (`DpaEvent_None`).

### 6.15.1 Enumerate Peripherals

This DPA request is called as a part of standard peripheral enumeration.

The purposes of request are:

1. Specify how many user peripherals are implemented.
2. If any standard peripheral is handled by custom DPA handler instead of default handler (overriding standard peripheral).
3. Specify HW profile and its version if one is implemented.

**Example**

```
case DpaEvent_None:
  if (IsDpaEnumPeripheralsRequest())
  {
    // One user peripheral defined
    _DpaMessage.EnumPeripheralsAnswer.UserPerNr = 1;
    // We override standard EEEPROM peripheral
    _DpaMessage.EnumPeripheralsAnswer.DefaultPer[PNUM_EEEPROM/8] |= 1 << (PNUM_EEEPROM % 8);
    // HW profile and version
    _DpaMessage.EnumPeripheralsAnswer.HwProfile = 0xABCD;
    _DpaMessage.EnumPeripheralsAnswer.HwProfileVersion = 0x1234;

    return TRUE;
  }
```

### 6.15.2 Get Peripheral Info

If the user code handles user or overriddes standard peripherals then this request is used to return information about the peripheral in the standard DPA format. If the handler does not handle the DPA “Get peripheral info request” then it must return FALSE to indicated error, otherwise it must return TRUE.

**Example**

```
case DpaEvent_None:
...
else if ( IsDpaPeripheralInfoRequest() )
{
  // 1st user peripheral
  if ( _PNum == PNUM_USER )
  {
      _DpaMessage.PeripheralInfoAnswer.PerT = PERIPHERAL_TYPE_LED;
      _DpaMessage.PeripheralInfoAnswer.PerTE = PERIPHERAL_TYPE_EXTENDED_READ_WRITE;
      _DpaMessage.PeripheralInfoAnswer.Par1 = LED_COLOR_UNKNOWN;
      goto DpaHandleReturnTRUE;
  }
  return TRUE;
}
```

### 6.15.3 Handle Peripheral Request

This request is sent whenever there is DPA request for a peripheral that was not handled by the default DPA code. Typically the code handles requests for user peripherals or overridden standard peripherals. If the handler does not handle the DPA request then it must return FALSE to indicated error, otherwise it must return TRUE.

Please note in the following code how to return an error state. Set `PNum` to `PNUM_ERROR_FLAG`, set 1. data byte of the DPA response to the error code, set 2. byte to the original `PNum` and finally specify that the length of the data being equal to 2. The best way is to use predefined union member at `_DpaMessage.ErrorAnswer`.

**Example**

```
case DpaEvent_None:
...
else if (IsDpaPeripheralInfoRequest())
  // ...
else
{
  // 1st user peripheral
  if (_PNum == PNUM_USER)
  {
      // Test for some data sent
      if (DpaDataLength == 0)
      {
        // Return error ERROR_DATA_LEN
        _DpaMessage.ErrorAnswer.ErrN = ERROR_DATA_LEN;
UserErrorAnswer:
        _DpaMessage.ErrorAnswer.PNumOriginal = _PNum;
        _PNum = PNUM_ERROR_FLAG;
        _DpaDataLength = sizeof(_DpaMessage.ErrorAnswer);
        return TRUE;
      }
      if (_PCmd == 0)
      {
        UseDataCmd0(_DpaMessage.Request.PData[0]);
        _DpaDataLength = 0;
        return TRUE;
      }
      else if (_PCmd == 1)
      {
        UseDataCmd1(_DpaMessage.Request.PData[0]);
        _DpaMessage.Response.PData[0] = someDataToReturn;
        _DpaDataLength = 1;
        return TRUE;
      }
      else
      {
        // Return error ERROR_PCMD
        _DpaMessage.ErrorAnswer.ErrN = ERROR_PCMD;
        goto UserErrorAnswer;
      }
  }
  return TRUE;
}
```

## 6.16 DPA API

The following functions can be called inside the Custom DPA Handler routine. Please note that after calling an API function the value of macro `GetDpaEvent()` is undefined.

### 6.16.1 DpaApiRfTxDpaPacket

```
void DpaApiRfTxDpaPacket(uns8 value, uns8 netDepth)
```

Available at [N] and [NC] devices. This function wraps all necessary code to send an RF DPA message. There are only a few global parameters or variables that have to be filled in before the call (see example below). Many other parameters are handled inside the function automatically. The following example shows a typical usage.

Meaning of the parameter `value` depends whether the message is sent from a coordinator or from a node.

- From Coordinator to Node: `value` specifies an exact number of hops used to return a DPA response from the node. IQRF OS function *optimizeHops* can be used to compute this value.
- From Node to Coordinator: `value` specifies a DpaValue that is returned with every DPA response.
  - If the coodinator is addressed by `COORDINATOR_ADDRESS = 0x00`, then the DPA packet is sent by the addressed coordinator to the interface master in case of [C] device or to the higher network by Bridge command in case of [NC] device after it is received.
  - If the coodinator is addressed by `LOCAL_ADDRESS = 0xfc`, then the DPA packet (request) is executed locally at the coordinator device.

Meaning of the parameter `netDepth` depends whether the message is sent from a coordinator or from a node. At both cases it is used to track the depth of the message when bridged among networks. When message is bridged to the lower network, the value is increased. When message is bridged (back - in case of DPA response) to the higher network the value is first decreased and then the actual bridging is performed only when the result is not zero. This ensures that the DPA response is not bridged “above” the sender of the original DPA request.

- From Coordinator to Node: use value 1.
  - From Node to Coordinator: use value 1 if the message should be terminated at the subordinate coordinator, use value 2 if the message should be terminated at the DPA interface of the same coordinator or at the coordinator above the same coordinator, etc.

Calling *DpaApiRfTxDpaPacket* is allowed only at Idle and AfterRouting events. The function does not take into account any IQMESH timing requirements, e.g. waiting for end of the routing process).

**Example**

```
// Generate new packet ID
PID = pid++;
// Number of hops = my VRN
RTHOPS = ntwVRN;
// No DPA Params used
_DpaParams = 0;
// Execute DPA request at coordinator
_NAdr = LOCAL_ADDRESS;
_NAdrHigh = 0;
// We will use LED peripheral
_PNum = PNUM_LEDR;
// Pulse the LED
_PCmd = CMD_LED_PULSE;
// HW profile
_HwProfile = 0x1234;
// Length of the data inside DPA request message
_DpaDataLength = 0;
// Transmit DPA message with DPA Value equal to the lastRSSI (can be any other value)
DpaApiRfTxDpaPacket(lastRSSI, 1);
```

### 6.16.2 DpaApiReadConfigByte

```
uns8 DpaApiReadConfigByte(uns8 index)
```

This function returns HWP configuration value from a given index (address).

**Example**

```
setRFchannel(DpaApiReadConfigByte(CFGIND_OS_CHANNEL_2ND));
```

### 6.16.3 DpaApiSendToIFaceMaster

```
void  DpaApiSendToIFaceMaster()
```

Available at [C] device. The function passes prepared DPA packet (response) to the interface master. If the interface master was not previously detected, then the call is actually ignored. If there is some older data at the interface bus not being collected by the interface master yet then the function waits until the data is read.
Calling *DpaApiSendToIFaceMaster* is allowed only at Idle, IFaceReceive and ReceiveDpaResponse events.

### 6.16.4 DpaApiRfTxDpaPacketCoordinator

```
void  DpaApiRfTxDpaPacketCoordinator()
```

Available at [C] device. This function is specially prepared for sending DPA requests from [C] to the [N] or [NC] in its network. It prepares even more of the requested parameters automatically compared to the DpaApiRfTxDpaPacket function. Last but not least it also takes care of waiting until routing of the previously sent (and received) packet is finished thus minimizing the probability of the network collision.
Calling *DpaApiRfTxDpaPacketCoordinator* is allowed only at Idle, AfterRouting and IFaceReceive events.

**Example**

```
case DpaEvent_Idle:
  {
// The following block of code demonstrates autonomous once per 60 s sending of packets
// if the {C} is not connected to the interface master
      if (IFacemasterNotConnected && DpaTicks.15 != 0)
      {
        // Setup new timer
        GIE = 0;
        DpaTicks = 60 * 100L;
        GIE = 1;

        // DPA request is broadcasted
        _NAdr = BROADCAST_ADDRESS;
        _NAdrHigh = 0;
        // Use red LED
        _PNum = PNUM_LEDR;
        // Make a LED pulse
        _PCmd = CMD_LED_PULSE;
        // HW profile
        _HwProfile = HWProfile_DoNotCheck;
        // This DPA request has no data
        _DpaDataLength = 0;
        // Send the DPA request
        DpaApiRfTxDpaPacketCoordinator();
      }

      return TRUE;
  }
```

### 6.16.5 DpaApiLocalRequest

```
void  DpaApiLocalRequest()
```

Performs a local DPA request. After the function returns a corresponding DPA response is available except when the original DPA request was a Batch. Calling *DpaApiLocalRequest* is allowed at Init, Idle, AfterRouting, BeforeSleep, AfterSleep and DisableInterrupts events. When a processed DPA message is not destroyed or used later then the function can be carefully used at ReceiveDpaResponse, IFaceReceive, ReceiveDpaRequest and BeforeSendingDpaResponse events too. To avoid reentrancy no Custom DPA Handler events (except Interrupt event) are called during local DPA request processing.

**Example**

```
case DpaEvent_Idle:
  if ( IsSleepTime() )
  {
    // Prepare OS Sleep DPA Request
    _PNum = PNUM_OS;
    _PCmd = CMD_OS_SLEEP;
    _DpaMessage.PerOSSleep.Time = 123;
    _DpaMessage.PerOSSleep.Control = 0b010;
    _DpaDataLength = sizeof( _DpaMessage.PerOSSleep );
    // Perform local DPA Request
    DpaApiLocalRequest();
    // If no error, pulse the LEDR after wake up
    if ( _PNum != PNUM_ERROR_FLAG )
      pulseLEDR();
  }
  return TRUE;
```

## 6.17 DPA API Variables

The following variables can be used within custom DPA handler routine. The variables marked by *[readonly]* are read-only variables. Writing to these variables will cause incorrect device behavior.

### 6.17.1 *bit* ProvidesRemoteBonding

*[readonly]* Equals to 1 when device provides remote pre-bonding, see Enable remote bonding.

### 6.17.2 *bit* RemoteBondingDone

*[readonly]* Equals to 1 when device provided pre-bonding to a new node.

### 6.17.3 *bit* IFacemasterNotConnected

*[readonly]* Valid at [C] device. Equals to 1 when master interface device was not connected during device startup process. Please note that this flag might become equal to 0 when a master interface device sends some data to the [C] device later.

### 6.17.4 *bit* NodeWasBonded

Valid at [N] and [NC] devices. Is set to 1 during Device startup process if the node was newly bonded. It is a programmer's responsibility to set this variable if default bonding mechanism is overridden at Reset event.

### 6.17.5 *bit* EnableIFaceNotificationOnRead

Valid at [N] and [NC] devices. Setting to 1 enables sending DPA notification to the interface master even in case od “read only” DPA request. Default value is 0.

### 6.17.6 *uns16* DpaTicks

Implemented at [C] device only. The value of this variable is decremented every 10 ms after Init event. The variable can be used for implementation of timing algorithms. As this 2 byte wide variable is modified internally within CPU interrupt routine the whole (both 2 bytes) variable should be accessed (both read or written) only when interrupt is disabled to ensure an atomic access.

**Example**

```
case DpaEvent_Idle:
      // Is timeout over?
      if ( DpaTicks.15 != 0 )
      {
            // Setup new 10s timeout
            GIE = 0;
            DpaTicks = 10 * 100L;
            GIE = 1;
...
```

### 6.17.7 *uns8* LP_XLP_touRF

Valid at [N] and [NC] devices and LP or XLP modes only. Timeout when receiving RF packets at LP or XLP modes. After a device startup it is filled with a respective value from HWP Configuration at index 0x0A. See that chapter for more details.

### 6.17.8 *uns8* ResetType

Identifies type of reset (stored at UserReg0 upon module reset). See Reset chapter at IQRF User's Guide for more information.

# 7 Device startup process

When device boots it first optionally goes into RFPGM mode supposed this mode is (enabled) configured in to OS tab of the TR Configuration dialog box at IQRF IDE. RFPGM mode is indicated by a repeated long green LED light followed by short red LED flash. RFPGM mode is terminated depending on its configuration in the IQRF IDE. RFPGM mode that is fully controlled by IQRF OS.

At the very begining it is possible to remotely start the device in so called *DPA Service Mode*. A special tool e.g. CATS - DPA Service Tool from IQRF IDE is needed to do it. In the DPA Service Mode the device can be fully controlled by individual DPA commands regardless of the device configuration so it gives possibility to update or fix a corrupted device configration, find out its network address, (un)bond it, find out OS information etc.

Then in case of full DPA version comes the check whether the device is IQRF Data Controlled Transceivers (DCTR). If this is not the case the program execution stops and both red and green LEDs flash rapidly.

Bonding or unbonding phase being valid only for [N] and [NC] devices comes next.

By default a bonding or a bond removal at node side is initiated and controlled by „default“ IQRF button connected to `PORTA.5` MCU pin which is normally available at IQRF development tools. Default behavior can be modified by an implementation of Reset event.

If node is not bonded then its red LED rapidly flashes (four times per second). Node waits for the button press. If the button is not pressed within 10s then the node goes into power saving sleep mode and red LED stops flashing. From the sleep mode the node can be woken up by the button press.

By pressing the button a bonding process is initiated. If the button is pressed the node continuously requests bonding (indicated by red LED). If the red LED becomes off and a green LED is lit when button is still pressed then the node is bonded. If the red LED keeps flashing rapidly after the button is released then the node is not bonded yet and the whole bonding phase repeats.

Already bonded node can be unbonded by the following procedure. Power off the node. Keep pressed the button and power up the node. Skip optional RFPGM mode depending on its configuration (typically pressed button terminates it). Keep button pressed. Green LED is then on. After 2 seconds the green LED goes off. Release the button immediately within 0.5 s. Unbonding is then confirmed by red LED being on for 1 second and consequently by the rapid red flashes described above. Such complicated unbonding procedure is needed in order to prevent unwanted unbonding caused by accidental button press after the device is reset.

At this point [N] an [NC] devices are bonded and ready to work in the DPA environment. This is indicated by short red LED flash. If the device has a temporary network address (0xFE) obtained by remote bonding then the device flashes twice.

Devices [C] and [NC] perform one green LED flash when they are ready. In case of [NC] device this flash goes together with 1st red LED flash.

Additionally [C] device checks a presence of the connected interface master device during startup. The [C] device (being always interface slave) sends the following asynchronous “Reset” DPA response equal (except PCmd) to Peripheral enumeration response to the interface master.

| NAdr | PNum | PCmd | HwProfile | PData |
|---|---|---|---|---|
| 0 | 0xFF | 0x3F | ? | See DPA response of Peripheral enumeration |

If the data are not collected from the interface by the interface master within 100 ms than the device consider interface master not being present. When interface master is not connected an extra green LED flash is carried out and API variable bit IFacemasterNotConnected is set to 1.

# 8 Constants

## 8.1 Peripheral numbers

```
#define  PNUM_COORDINATOR    0x00
#define  PNUM_NODE           0x01
#define  PNUM_OS             0x02
#define  PNUM_EEPROM         0x03
#define  PNUM_EEEPROM        0x04
#define  PNUM_RAM            0x05
#define  PNUM_LEDR           0x06
#define  PNUM_LEDG           0x07
#define  PNUM_SPI            0x08
#define  PNUM_IO             0x09
#define  PNUM_THERMOMETER    0x0A
#define  PNUM_PWM            0x0B
#define  PNUM_UART           0x0C
#define  PNUM_FRC            0x0D

#define  PNUM_USER           0x20

#define  PNUM_ERROR_FLAG     0xFE
```

## 8.2 Response Code

```
STATUS_NO_ERROR =             0,  // No error
ERROR_FAIL =                  1,  // General fail
ERROR_PCMD =                  2,  // Incorrect PCmd
ERROR_PNUM =                  3,  // Incorrect PNum or PCmd
ERROR_ADDR =                  4,  // Incorrect Address
ERROR_DATA_LEN =              5,  // Incorrect Data length
ERROR_DATA =                  6,  // Incorrect Data
ERROR_HWPROFILE =             7,  // Incorrect HW Profile type used
ERROR_NADR =                  8,  // Incorrect NAdr
ERROR_IFACE_CUSTOM_HANDLER = 9,  // Data from interface consumed by Custom DPA Handler

STATUS_CONFIRMATION =      0xFF  // Error code used to mark confirmation
```

## 8.3 DPA Commands

```
#define     CMD_COORDINATOR_ADDR_INFO  0
#define     CMD_COORDINATOR_DISCOVERED_DEVICES 1
#define     CMD_COORDINATOR_BONDED_DEVICES 2
#define     CMD_COORDINATOR_CLEAR_ALL_BONDS 3
#define     CMD_COORDINATOR_BOND_NODE 4
#define     CMD_COORDINATOR_REMOVE_BOND 5
#define     CMD_COORDINATOR_REBOND_NODE 6
#define     CMD_COORDINATOR_DISCOVERY 7
#define     CMD_COORDINATOR_SET_DPAPARAMS 8
#define     CMD_COORDINATOR_SET_HOPS 9
#define     CMD_COORDINATOR_DISCOVERY_DATA 10
#define     CMD_COORDINATOR_BACKUP 11
#define     CMD_COORDINATOR_RESTORE 12
#define     CMD_COORDINATOR_READ_REMOTELY_BONDED_MID 15
#define     CMD_COORDINATOR_CLEAR_REMOTELY_BONDED_MID 16
#define     CMD_COORDINATOR_ENABLE_REMOTE_BONDING 17

#define     CMD_NODE_READ 0
#define     CMD_NODE_REMOVE_BOND 1
#define     CMD_NODE_READ_REMOTELY_BONDED_MID 2
#define     CMD_NODE_CLEAR_REMOTELY_BONDED_MID 3
#define     CMD_NODE_ENABLE_REMOTE_BONDING 4
#define     CMD_NODE_REMOVE_BOND_ADDRESS 5
#define     CMD_NODE_BACKUP 6
#define     CMD_NODE_RESTORE 7

#define     CMD_OS_READ 0
#define     CMD_OS_RESET 1
#define     CMD_OS_READ_CFG 2
#define     CMD_OS_RFPGM 3
#define     CMD_OS_SLEEP 4
#define     CMD_OS_BATCH 5
#define     CMD_OS_SET_USEC 6
#define     CMD_OS_SET_MID 7

#define     CMD_RAM_READ 0
#define     CMD_RAM_WRITE 1

#define     CMD_EEPROM_READ CMD_RAM_READ
#define     CMD_EEPROM_WRITE CMD_RAM_WRITE

#define     CMD_EEEPROM_READ CMD_RAM_READ
#define     CMD_EEEPROM_WRITE CMD_RAM_WRITE

#define     CMD_LED_SET_OFF 0
#define     CMD_LED_SET_ON 1
#define     CMD_LED_GET 2
#define     CMD_LED_PULSE 3

#define     CMD_SPI_WRITE_READ 0

#define     CMD_IO_DIRECTION  0
#define     CMD_IO_SET 1
#define     CMD_IO_GET 2

#define     CMD_THERMOMETER_READ 0

#define     CMD_PWM_SET 0

#define     CMD_UART_OPEN 0
#define     CMD_UART_CLOSE 1
#define     CMD_UART_WRITE_READ 2

#define     CMD_FRC_SEND 0
#define     CMD_FRC_EXTRARESULT 1

#define     CMD_GET_PER_INFO  0x3f
```

## 8.4 Peripheral Types

```
PERIPHERAL_TYPE_DUMMY = 0x00,
PERIPHERAL_TYPE_COORDINATOR = 0x01,
PERIPHERAL_TYPE_NODE = 0x02,
PERIPHERAL_TYPE_OS = 0x03,
PERIPHERAL_TYPE_EEPROM = 0x04,
PERIPHERAL_TYPE_BLOCK_EEPROM = 0x05,
PERIPHERAL_TYPE_RAM = 0x06,
PERIPHERAL_TYPE_LED = 0x07,
PERIPHERAL_TYPE_SPI = 0x08,
PERIPHERAL_TYPE_IO = 0x09,
PERIPHERAL_TYPE_UART = 0x0a,
PERIPHERAL_TYPE_THERMOMETER = 0x0b,
PERIPHERAL_TYPE_ADC = 0x0c, (*)
PERIPHERAL_TYPE_PWM = 0x0d,
PERIPHERAL_TYPE_FRC = 0x0e,
PERIPHERAL_TYPE_USER_AREA = 0x80,
```

(*) Peripheral of this type not defined and implemented yet

## 8.5 Extended Peripheral Characteristic

```
PERIPHERAL_TYPE_EXTENDED_DEFAULT = 0b00,
PERIPHERAL_TYPE_EXTENDED_READ = 0b01,
PERIPHERAL_TYPE_EXTENDED_WRITE = 0b10,
PERIPHERAL_TYPE_EXTENDED_READ_WRITE = PERIPHERAL_TYPE_EXTENDED_READ |
  PERIPHERAL_TYPE_EXTENDED_WRITE
```

## 8.6 HW Profiles

```
HWProfile_None = 0,               // No HW Profile implemented
HWProfile_UserArea = 0x0101,      // User HW Profile type area
HWProfile_ReservedArea = 0xf000,  // Reserved HW Profile type area
HWProfile_Reserved = 0xfffe,      // Reserved HW Profile type
HWProfile_DoNotCheck = 0xffff     // Use this type to override HW Profile check
```

## 8.7 LED_COLOR

```
LED_COLOR_RED = 0,
LED_COLOR_GREEN = 1,
LED_COLOR_BLUE = 2,
LED_COLOR_YELLOW = 3,
LED_COLOR_WHITE = 4,
LED_COLOR_UNKNOWN = 0xff
```

# 9 Migration notes

## 9.1 From DPA 1.00 to DPA 2.00

- Every DPA Request/Response contains new 2B HwProfile parameter, see General message parameters.
- Changes of parameters or response results of the followingcommands, services or API: `CMD_COORDINATOR_DISCOVERY`, `CMD_COORDINATOR_BACKUP`, `CMD_COORDINATOR_RESTORE`, `CMD_NODE_ENABLE_REMOTE_BONDING`, `CMD_NODE_READ`, `CMD_OS_READ_CFG`, `CMD_OS_READ`, `CMD_OS_BATCH`, `CMD_UART_OPEN`, `Peripheral enumeration`, `Autoexec`, `DpaApiRfTxDpaPacket`.
- [C] device sends „Reset“ message upon startup, see Device startup process.
- Notification event called even after read-only DPA response.
- Custom DPA Handler location and reserved Flash memory size changed and events renumbered. Custom DPA Handler must be recompiled and uploaded.
- Custom DPA Handler must use `case DpaEvent_None:` instead of `default:`
- Event `DpaEvent_Async` renamed to `DpaEvent_AfterRouting`.
- Node can address the coordinator by `COORDINATOR_ADDRESS` or `LOCAL_ADDRESS`.
- Changed LED indication of the forbidden address upon Node startup at demo mode.
- LED peripherals are not limited to demo version only.

# 10 Document revision

- 140604 For DPA v2.01.
- 130513 First release for DPA v2.00

# Sales and Service

## Corporate office

IQRF Alliance s.r.o., Prumyslova 1275, 506 01 Jicin, Czech Republic, EU
Tel: +420 493 538 125, Fax: +420 493 538 126, www.iqrfalliance.org

## Partners and distribution

Please visit www.iqrf.org/partners.

## Trademarks

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

## Legal

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

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

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

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