IQRF DPA Framework

Technical guide

Version v1.00

For IQRF OS v3.04D

Smarter Wireless. Simply.
IQRF ALLIANCE

Contents

1 Introduction

DPA protocol is a simple byte oriented protocol used to control services and peripherals of IQMESH network devices (coordinator and nodes) via SPI interface. The protocol is interfaced via SPI interface 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.

DPA protocol implementation is distributed in the form of IQRF plug-in. There is a demo version available having the following features:

- Maximum node network address is 5. Demo node device having unsupported address flashes periodically red and green LEDs after reset. Demo coordinator does not allow to address, to bond and to rebond node with an unsupported address.
- In Demo version a Custom DPA handler is not called.
- Discovery process is indicated by LEDs flashing.

2 Basics

DPA protocol uses byte structured messages to communicate at IQMESH network. Every message always contains three mandatory parameters NAdr, PNum, PCmd (*triplet* from now). The message can optionally hold data (array of bytes) to be transmitted or received. They are always described next to the triplet throughout this document. Although triplet 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 triplet 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 General message parameters

Parameter	Value [hex]		Description
NAdr [2B]	00 01-EF F0-FB FC FD-FE FF 100-FFFF	IQMESH Coordinator IQMESH Node address Reserved Local (over SPI) device Reserved IQMESH Broadcast address Reserved for 2 byte address	Network device address. Although it is 2 bytes wide, the 2B addressing is not supported yet. NAdr 2 bytes are coded using little-endian style.
PNum [1B]	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 20-6F 70-FF	COORDINATOR NODE OS EEPROM EEEPROM RAM LEDR * LEDG * SPI IO Thermometer PWM ** UART User peripherals Reserved	Peripheral number (0x00 – 0x1F reserved for standard peripherals) 1st user peripheral is always 0x20, 2nd is 0x21 etc.
PCmd [1B]	0-3E 3F	Reserved	Command specifying an action to be taken. Actual allowed value range depends on the peripheral type. The most significant bit indicates DPA response message.

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

2.3 Communication flow

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

- SPI master sends **DPA request**
- If addressee (NAdr) is a (remote) IQMESH Node, not a local over the SPI connected device (applies only to coordinator)
  - SPI slave immediately sends **DPA confirmation** back to the SPI master
  - Node processes the DPA message
  - If the DPA message does not have a read side-effect and the SPI 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 SPI master
- In case of a local device
  - Device processes the DPA message
  - Device returns **DPA response** back to SPI master

2.3.1 DPA request

DPA request consists of *triplet* with optional data, depending on the actual request.

2.3.2 DPA confirmation

DPA confirmation confirms a reception of DPA request by SPI slave to SPI master. It consists of the same *triplet* 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):

Data length [B]	Timeslot length [ms]
< 22	30
22 - 42	40
> 42	50

This knowledge can be used to implement a precise timing of the control system connected to the coordinator device via SPI 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 SPI interface is ready (SPI status is `ReadyCommunication`) and no data remained for reading at SPI interface.
3. Send DPA Request via SPI interface.
4. Receive DPA Confirmation via SPI. 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 SPI 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 (5 = 50 ms). 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).

2.3.3 DPA notification

DPA notification notifies a connected SPI master at the node side that there was a DPA request without a read side-effect processed by the node. It consists of the same *triplet* that was part of the original DPA request except NAdr stores address of the sender, not addressee. DPA notification is therefore always 4 B long.

2.3.4 DPA response

DPA response is an actual answer to the DPA request. DPA response consists of the same *triplet* that was part of the original DPA request except the response message is flagged by the most significant bit of PCmd. 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.

2.4 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`
- **DPA response** (slave → master)
  `NAdr=0x0000, PNum=0x06, PCmd=0x81, 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, Data={0x01}`(Address), `{0x02}`(Length)
- **DPA response** (slave → master)
  `NAdr=0x00FC, PNum=0x05, PCmd=0x80`
  `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`
- **DPA confirmation** (slave → master)
  `NAdr=0x000A, PNum=0x07, PCmd=0x01, 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, 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

2.5 Device exploration

2.5.1 Peripheral enumeration

Request

NAdr	PNum	PCmd
NAdr	0xFF	0x3F

Response

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

DpaVer — DPA protocol version
- 1st byte: bits 0-6 = minor version, bit 7 = demo version
- 2nd byte: major version

PerNr — Number of user defined peripherals
StdfPers — Bits array (starting from LSb of the 1st byte) specifying presence of each one of 32 standard peripherals
HwProf — Hardware profile type, coded using little-endian style, 0x0000 if not present,
HwProfV — Hardware profile version, 1st byte = minor version, 2nd byte = major version

Example

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

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 IO0 (C1 pin)
- 0x0A IO1 (C2 pin)

bit array: `11100110.00000110.00000000.00000000`

2.5.2 Get peripheral information

Request

NAdr	PNum	PCmd
NAdr	PNum	0x3F

Response

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

PerT — Peripheral type
PerTE — Extended peripheral characteristic
Par1 — Optional peripheral specific information
Par2 — Optional peripheral specific information

2.5.3 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
NAdr	0xFF	Per

Per First peripheral from the list to get the information about

Response

NAdr	PNum	PCmd	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 does not exist at index x, then $PerT_x$ = `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 sent using broadcast address.

3.1.1 Writing to peripheral in general

Request

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

n Data length

Response

NAdr	PNum	PCmd	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
NAdr	PNum	PCmd

Response

NAdr	PNum	PCmd	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 = 0

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 bytes that can be sent in the DPA messages
Par2 Not used

3.2.2 Get addressing information

Returns basic network information.

Request

NAdr	PNum	PCmd
NAdr	0	0x00

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0	1
NAdr	0	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
NAdr	0	0x02

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0	…	31
NAdr	0	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
NAdr	0	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. 10s blocking delay when this function is called.

Request

NAdr	PNum	PCmd	0	1
NAdr	0	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	ErrN	DpaValue	0	1
NAdr	0	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	0
NAdr	0	0x05	BondAdr

BondAdr Address of the node to remove the bond to

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0
NAdr	0	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	0
NAdr	0	0x06	BondAdr

BondAdr Address of the node to be re-bonded

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0
NAdr	0	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	0	1
NAdr	0	0x07	TxPower	Zones

TxPower TX Power used for discovery.
Zones Specifies maximum number of zones that will be created during discovery process. If value 0 is used, then actually number of bonded devices will be used.

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0
NAdr	0	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 *
	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	0
NAdr	0	0x08	DPA Param

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0
NAdr	0	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	0	1
NAdr	0	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	ErrN	DpaValue	0	1
NAdr	0	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	0
NAdr	0	0x0A	Address

Address Address of the discovery data.

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0	…	15
NAdr	0	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. Data structure is not public.

Request

NAdr	PNum	PCmd	0
NAdr	0	0x0B	Index

Index Index of the block of data

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0	…	17
NAdr	0	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 via Backup commands must be written to the device.

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

- 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.
- Discovery command is executed at the restored coordinator before its 1st network use.

Request

NAdr	PNum	PCmd	0	…	17
NAdr	0	0x0C	Network data		

Network data One block of the coordinator network info data previously obtained via 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	0	1	2
NAdr	0	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 abtained by calling Read remotely bonded module ID.

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

3.3 IQMESH – Node

PNum = 1

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 bytes that can be sent in the DPA messages
Par2 Not used

3.3.2 Read

Returns IQMESH specific node information.

Request

NAdr	PNum	PCmd
NAdr	1	0x00

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0	1	2	3
NAdr	1	0x80	0	?	Vrn	ParentVrn	ZoneIndex	Address

Vrn VRN (Virtual Routing Number)
ParentVrn Parent VRN
ZoneIndex Zone index (Zone number + 1)
Address Logical network address of the node

3.3.3 Remove bond

The bond is marked as unbonded (removed from network) using `removeBond()` IQRF call. The node will actually keep working in the network until next reset. Bonding state of the node at the coordinator side is not effected at all.

Request

NAdr	PNum	PCmd
NAdr	1	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 againg, 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.

Request

NAdr	PNum	PCmd	0	1
NAdr	1	0x04	Bonding mask	Control

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.

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.

Request

NAdr	PNum	PCmd
NAdr	1	0x02

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0	1	2	3	4	5
NAdr	1	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
NAdr	1	0x03

Response

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

3.4 OS

PNum = 2

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 interesting system information about the node.

Request

NAdr	PNum	PCmd
NAdr	2	0x00

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0 - 3	4	5	6-7	8	9
NAdr	2	0x80	0	?	ModuleID	OSVersion	McuType	OsBuild	Rssi	SupplyVoltage

ModuleID, OSVersion, McuType, OsBuild — See `moduleInfo()` at IQRF OS Reference Guide
Rssi — See `lastRSSI` at IQRF Reference Guide
SupplyVoltage — See `getSupplyVoltage()` at IQRF Reference Guide

3.4.3 Reset

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

Request

NAdr	PNum	PCmd
NAdr	2	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.

Request

NAdr	PNum	PCmd
NAdr	2	0x02

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0	…	n
NAdr	PNum	0x82	0	?	Configuration memory		

Configuration memory Configuration memory data 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
NAdr	2	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	0	1	2
NAdr	2	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 132ms 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 via *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 to execute more individual DPA requests within one original DPA request. It is not allowed to embed Batch command itself within series of individual DPA requests.

Request

NAdr	PNum	PCmd	0	...	n
NAdr	2	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 set of DPA requests. It switches on red LED, then waits for 200 ms (using I/O peripheral) and finally switches the LED off.

```
NAdr=0x0001, PNum=0x02, PCmd=0x05, Data=
[1st command] {0x03(length), 0x06(PNum=LEDR), 0x01(PCmd=LED on)},
[2nd command] {0x06(length), 0x09(PNum=I/O), 0x01(PCmd=Set), 0xFF(Delay command), 0x00C8(200ms)}
[3rd command] {0x03(length), 0x06(PNum=LEDR), 0x00(PCmd=LED off)},
{0x00(end of batch)}
```

Response

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

3.5 EEPROM

PNum = 3

3.5.1 Peripheral information

PerT `PERIPHERAL_TYPE_EEPROM`
PerTE `PERIPHERAL_TYPE_EXTENDED_READ_WRITE`
Par1 Size in bytes
Par2 Maximum data block length

3.5.2 Read

Reads data from memory.

Request

NAdr	PNum	PCmd	0	1
NAdr	3	0x00	Addr	Len

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

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0	…	n-1
NAdr	3	0x80	0	?	$PData_0$	…	$PData_{n-1}$

n Data length

3.5.3 Write

Writes data to memory.

Request

NAdr	PNum	PCmd	0	1	2	3	…	n+3
NAdr	3	0x01	Addr	HwProfile		$PData_0$	…	$PData_{n-1}$

HwProfile HW Profile type. It is checked against a HW Profile that is implemented in the device. If it does not match, then `ERROR_HWPROFILE` is returned at ErrN. This word value is coded using little-endian style.
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 = 4

3.6.1 Peripheral information

PerT `PERIPHERAL_TYPE_EEPROM`
PerTE `PERIPHERAL_TYPE_EXTENDED_READ_WRITE`
Par1 Memory size in blocks (see Par2)
Par2 Data block size

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

3.7 RAM

PNum = 5

3.7.1 Peripheral information

PerT	PERIPHERAL_TYPE_RAM
PerTE	PERIPHERAL_TYPE_EXTENDED_READ_WRITE
Par1	Size in bytes
Par2	Maximum data block length

3.7.2 Read & Write

See EEPROM.

3.8 SPI (Slave)

PNum = 8

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.

3.9 LED

PNum = 6 or 7 for standard red respectively green LED.

The peripheral is available at Demo version only.

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
NAdr	6 or 7	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
NAdr	6 or 7	0x02

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0
NAdr	6 or 7	0x82	0	?	OnOff

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

3.10 IO

PNum = 9

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 via 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. See datasheet of the PIC MCU describing TRISx ports.

Request

NAdr	PNum	PCmd	0	1	2	…	n * 3	n * 3 + 1	n * 3 + 2
NAdr	9	0x00	$port_0$	$mask_0$	$value_0$		$port_n$	$mask_n$	$value_n$

port Specifies port to setup a direction to. 0=TRISA, 1=TRISB, …
mask Masks pins of the port.
value Actual direction bits for the masked pins. 0=output, 1=input.

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	0	1	2	…	n * 3	n * 3 + 1	n * 3 + 2
NAdr	9	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
NAdr	9	0x02

Response

NAdr	PNum	PCmd	ErrN	DpaValue	0	…	N
NAdr	9	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, 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, 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, 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, Data={00}`(No error)`, {0x07}`(DPA Value)

3.11 Thermometer

PNum = 10 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
NAdr	0x0A	0x00

Response

NAdr	PNum	PCmd	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 = 11 for standard MCU PWM peripheral

The peripheral is available at Demo version only and not at the Coordinator [C] device.

3.12.1 Peripheral information

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

3.12.2 Set

Sets PWM parameters.

Request

NAdr	PNum	PCmd	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, Data={0x02,0x7d,0x40}`
- **DPA response** (slave > master)
  `NAdr=0x0000, PNum=0x0B, PCmd=0x80, 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, Data={0x22,0x7d,0x57}`
- **DPA response** (slave > master)
  `NAdr=0x0000, PNum=0x0B, PCmd=0x80, Data={0x00}` (No error)

3.13 UART

PNum = 12 for standard UART peripheral

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

3.13.1 Peripheral information

PerT — `PERIPHERAL_TYPE_UART`

PerTE — `PERIPHERAL_TYPE_READ_WRITE`

Par1 — `PERIPHERAL_UART_MAX_DATA_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. MCU IO pin PORTC.5=SDO that is often connected to PORTC.7=Rx at many IQRF TR modules is set as input. If this is undesirable then use IO peripheral to set it back to output state.

Request

NAdr	PNum	PCmd	0
NAdr	0x0C	0x00	BaudRate

BaudRate specifies baud rate:

- 0x00 2400 baud
- 0x01 4800 baud
- 0x02 9600 baud
- 0x03 19200 baud
- 0x04 57600 baud
- 0x05 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, Data={0x02}(9600 baud)
- **DPA response** (slave > master)
  NAdr=0x0000, PNum=0x0C, PCmd=0x80, Data={0x00}(No error)

3.13.3 Close

Closes UART interface.

Request

NAdr	PNum	PCmd
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	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	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 2

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

- **DPA request** (master > slave)

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

  `NAdr=0x0000, PNum=0x0C, PCmd=0x82, Data={0x00}`(No error)

Example 3

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

- **DPA request** (master > slave)

  `NAdr=0x0000, PNum=0x0C, PCmd=0x02, Data={0x01}`(10ms timeout) `{0x00,0x01,0x02}`(written data)
- **DPA response** (slave > master)

  `NAdr=0x0000, PNum=0x0C, PCmd=0x82, Data={0x00}`(No error) `{0xaa,0xbb,x0cc,0xdd}`(read data)

4 HWP Configuration

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

Address	Description
1	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.
2	
3	
4	
5	Various DPA configuration flag bits:
bit 0	If set, then a Custom DPA handler is called in case of event. The handler can define user peripherals as well as handle messages to standard peripherals.
bit 1	If set, then DPA can be controlled by a local SPI interface. In this case SPI peripheral must not be enabled. This option is also not valid for a main network coordinator device.
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.
6	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.
7	Same as above but second B channel.
8	RF output power. Valid numbers 0-7.
9	RF signal filter. Valid numbers 0-64.
17	RF channel A of the main network. Valid numbers depend on used RF band.
18	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 via 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 SPI notification is not performed although DPA via SPI 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      Data
1. 0x03, 0x07, 0x01(On)
2. 0x04, 0x0C, 0x00(open), 0x02(9600 baud)
3. 0x09, 0x0C, 0x02(write), 0xff(no UART read), {0x01, 0x02, 0x03, 0x04, 0x05}(data)
4. 0x0b, 0x05, 0x01(write), 0x0a(address), {0xff, 0xff}(HW Profile), {0x06, 0x07, 0x08, 0x09, 0x0a}(data)
5. 0x00(end of Autoexec)
```

6 Custom DPA Handler

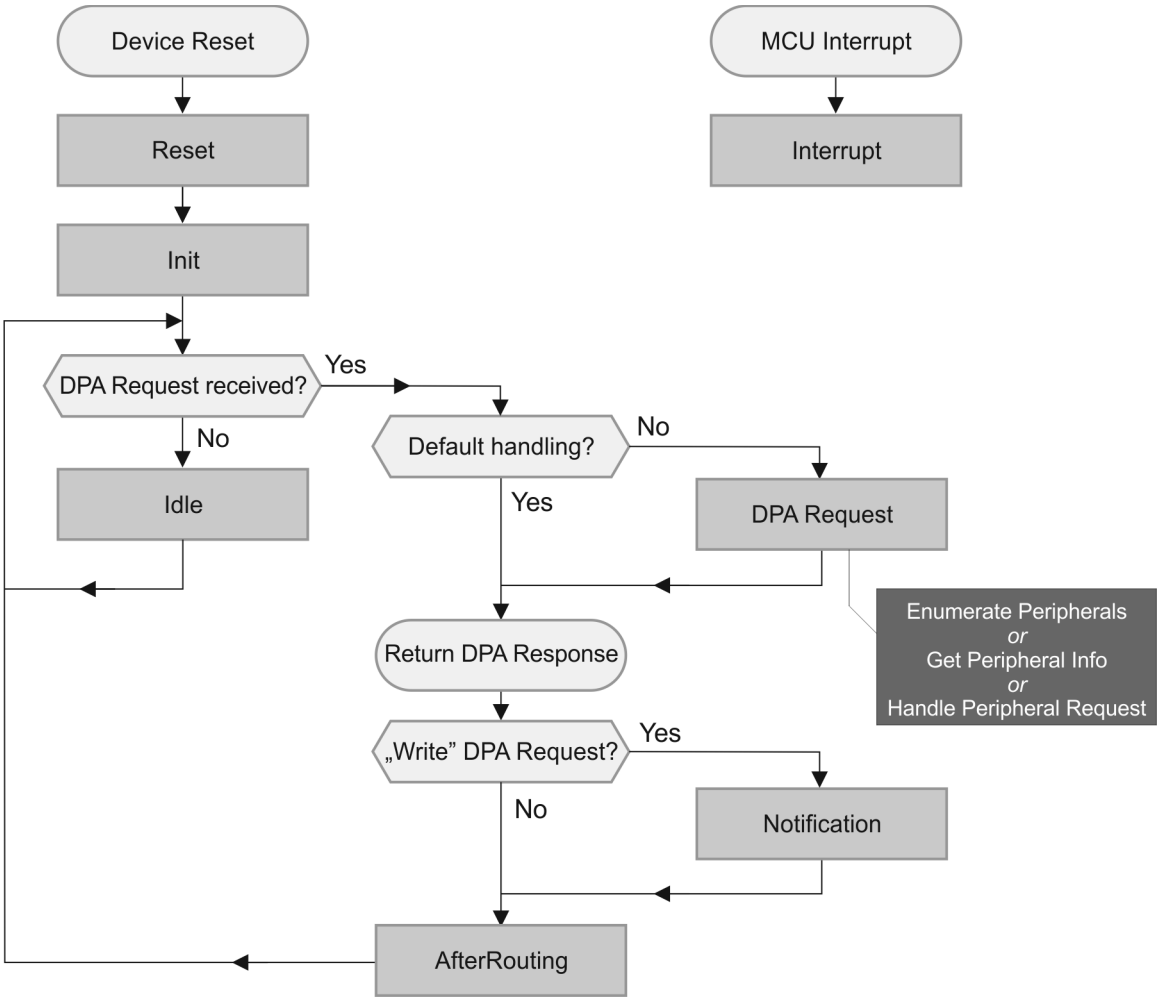
Custom DPA Handler is not available at Demo version.

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.

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. Variables as well as function parameters must be allocated in the standard RAM bank 11 only.
3. Do not use `bufferRF`, `bufferCOM`, `bufferINFO` and `bufferAUX` at all.
4. 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.
5. Special attention must be paid to the implementation of an Interrupt event. See details in the dedicated chapter.
6. Do not use timer TMR6 at the coordinator only device [C].
7. Do not use IQRF methods `start[Long]Delay()` and `waitDelay()`. Use `waitMS()` instead.

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.

6.1 Interrupt

This 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.
- 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 SPI) message to handle. The event is also used to obtain user DPA Value that is transferred via `DLEN` variable. The time spent handling this event is critical.

Example

```
case DpaEvent_Idle:
     userReg0 = myUserDpaValue;
     return TRUE;
```

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 ID. Immediately after the event is processed the Autoexec is executed. This event is typically used to initialize peripherals and global variables.

Example

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

6.4 Notification

This event is called when a DPA request was successfully processed and there is no data returned. Typically it means that a “write” DPA request was processed. User can sense what peripheral was written to 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 TRUE;
```

6.5 AfterRouting

[sync] This event is called after the DPA response was sent and (optional) Notification event and (optional) SPI Notification is sent. In any case the packet routing of the original DPA request is finished.

Example

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

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].

Example

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

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].

Example

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

6.8 Reset

This event is called just after the module was reset. It can be used to handle bonding/unbonding of the node. 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) that is passed during remote bonding process and can be read by Read remotely bonded module ID.

Example

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

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

6.9 Disable Interrupts

This event is called when device needs all hardware interrupts to be disabled. Such moment occurs e.g. Just before entering RFPGM mode.

Example

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

6.10 DPA Request

The following events handle original DPA requests.

6.10.1 Enumerate Peripherals

This event is called as a part of standard peripheral enumeration.

The purposes of this event 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 ID and Version of hardware profile if one is implemented.

Example

```
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 ID and version
  _DpaMessage.EnumPeripheralsAnswer.HwProfile = 0xABCD;
  _DpaMessage.EnumPeripheralsAnswer.HwProfileVersion = 0x1234;
  return TRUE;
}
```

6.10.2 Get Peripheral Info

If the user code handles user or overridden standard peripherals then this event 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

```
...
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;
}
```

6.10.3 Handle Peripheral Request

This event is called 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.

Example

```
else if (IsDpaPeripheralInfoRequest())
  // ...
else
{
  // 1st user peripheral
  if (_PNum == PNUM_USER)
  {
      // Test for some data sent
      if (!IsDpaPacketNoData())
      {
        _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
      {
        _DpaMessage.ErrorAnswer.ErrN = ERROR_PCMD;
        goto UserErrorAnswer;
      }
  }
  return TRUE;
}
```

6.11 DPA API

The following functions can be called inside the Custom DPA Handler routine.

6.11.1 DpaApiRfTxDpaPacket

```
void DpaApiRfTxDpaPacket(uns8 value)
```

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. 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.

Calling *DpaApiRfTxDpaPacket* is allowed only at Idle and AfterRouting events.

Example

```
// Packet ID
PID = 0;
// Number of hops is optimized
RTDT0 = 0xFF;
// No DPA Params used
_DpaParams = 0;
// Data is sent to the Coordinator
_NAdr = COORDINATOR_ADDRESS;
_NAdrHigh = 0;
// We will use LED peripheral
_PNum = PNUM_LEDR;
// Write command to the RAM
_PCmd = CMD_LED_SET_ON;
// 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);
```

6.11.2 DpaApiReadConfigByte

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

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

Example

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

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.
Bonding or unbonding phase valid only for [N] and [NC] devices comes after the previously described optional RFPGM mode that is fully controlled by IQRF OS.

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 behaviour can be modified by implementing 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 optinal RFPGM mode depending on ots 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 procesure 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.

8 Constants

8.1 Response Code

```
STATUS_NO_ERROR =       0,    // No error
ERROR_FAIL =            1,    // General fail
ERROR_PCMD =            2,    // Incorrect PCmd
ERROR_PNUM =            3,    // Incorrect PNum
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
STATUS_CONFIRMATION = 0xff  // Error code used to mark confirmation
```

8.2 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_NODE_READ 0
#define     CMD_NODE_REMOVE_BOND 1

#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_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_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_GET_PER_INFO  0x3f
```

8.3 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,
```

8.4 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
```

8.5 HW Profiles

```
  HWProfile_None = 0,               // No HW Profile implemented
  HWProfile_Reserved = 0xfffe,      // Reserved HW Profile type
  HWProfile_DoNotCheck = 0xffff     // Use this type at memory write message to override
                                    // HW Profile check
```

8.6 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 Document revision

- 130911 First release

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.iqrf.org/alliance.

Partners and distribution

Please visit www.iqrf.org/partners.

Trademarks

Legal

Smarter Wireless. Simply.
IQRF ALLIANCE