BLEPARK

USER MANUAL

V1.2.1

FCC & IC statements (1/2)

Caution

The user is cautioned that changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This device complies with Part 15 of the FCC Rules and Industry Canada licence-exempt RSS standard(s). Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

Le présent appareil est conforme aux CNR d’Industrie Canada applicables aux appareils radio exempts de licence. L’exploitation est autorisée aux deux conditions suivantes: (1) l’appareil ne doit pas produire de brouillage, et (2) l’utilisateur de l’appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d’en compromettre le fonctionnement.

FCC & IC statements (2/2)

NOTE

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

FCC and IC Radiation Exposure Statement:

This equipment complies with FCC and Canada radiation exposure limits set forth for an uncontrolled environment. This transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.

This equipment should be installed and operated with a minimum distance of 20cm between the radiator and your body.

Déclaration d'IC sur l'exposition aux radiations:

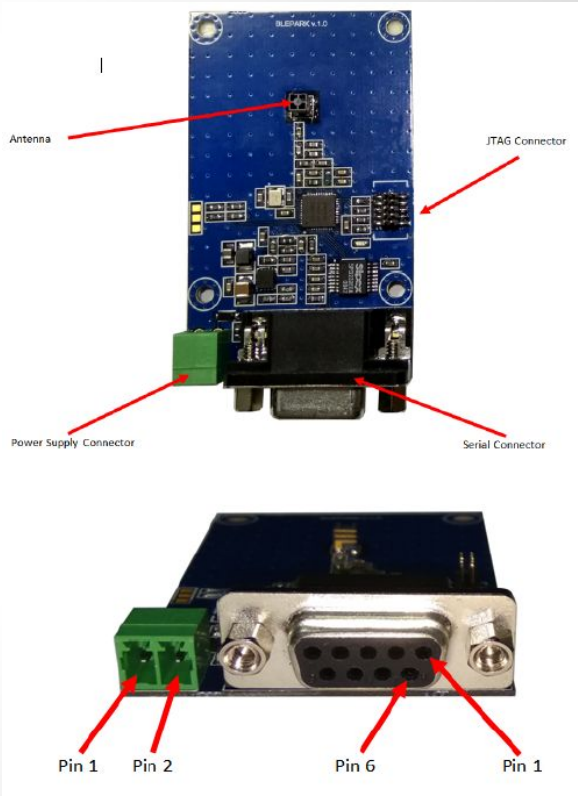
Cet équipement est conforme aux limites d'exposition aux radiations définies par le Canada pour des environnements non contrôlés. Cet émetteur ne doit pas être installé au même endroit ni utilisé avec une autre antenne ou un autre émetteur.

Cet équipement doit être installé et utilisé à une distance minimum de 20 cm entre l'antenne et votre corps.

HARDWARE DESCRIPTION

- Main components:
  - Chip antenna
  - Jtag connector for programming
  - Power supply header
  - Serial port connector
- Power Supply
  - Voltage range: +12 .. +24V DC
  - Current consumption: 10mA MAX

Power Supply Connector		
Pin 2	V +	Positive Power Supply
Pin 1	0V / Ground	Power Supply Return



SERIAL AND JTAG PINOUT

Serial Interface

RS-232 ; Baud rate: 57600; Stop Bits: 1; Parity: None

Serial Connector					
BLEPARK D-SUB 9 Female (DCE)			HOST D-SUB 9 Male (DTE)		
Pin 1	NA		Pin 1	DCD	Input
Pin 2	TXD	Output	Pin 2	RXD	Input
Pin 3	RXD	Input	Pin 3	TXD	Output
Pin 4	NA		Pin 4	DTR	Output
Pin 5	GND	Common	Pin 5	GND	Common
Pin 6	NA		Pin 6	DSR	Input
Pin 7	CTS	Input	Pin 7	RTS	Output
Pin 8	RTS	Output	Pin 8	CTS	Input
Pin 9	NA		Pin 9	RI	Input

JTAG

JTAG Connector	
Pin 1	Vref
Pin 2	SWDIO
Pin 3	GND
Pin 4	SWCLK
Pin 5	GND
Pin 6	NA
Pin 7	NA
Pin 8	NA
Pin 9	GND
Pin 10	NA

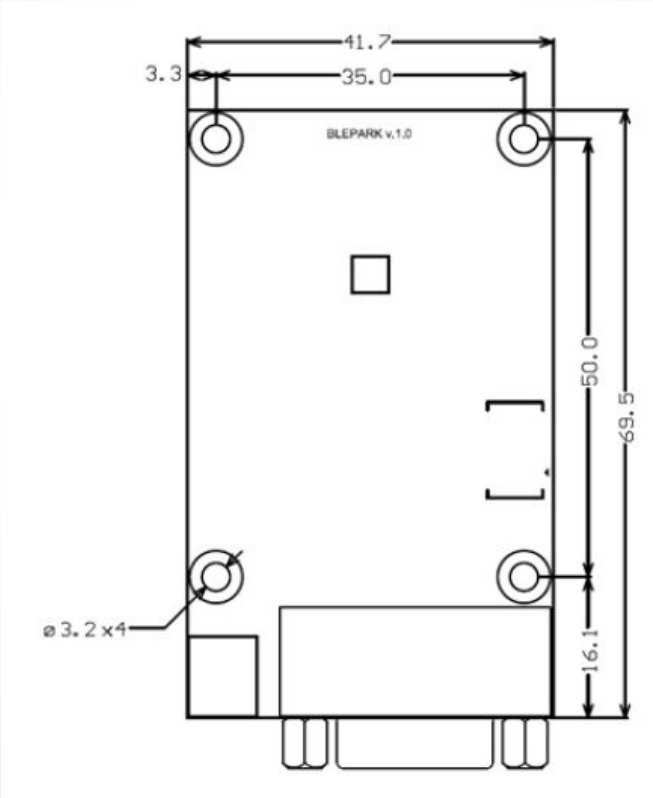
Chip characteristics

The board uses SoC nrf51422 from Nordic Semiconductor with integrated RF transceiver.

The following list gathers the main characteristics

- 2.4 GHz transceiver
- -90 dBm sensitivity in ANT mode
- -93 dBm sensitivity in Bluetooth® low energy mode
- 250 kbps, 1 Mbps, 2 Mbps supported data rates
- TX Power +4 dBm in 4 dB steps
- TX Power -30 dBm Whisper mode
- 13 mA peak RX, 10.5 mA peak TX (0 dBm)
- 9.7 mA peak RX, 8 mA peak TX (0 dBm) with DC/DC
- RSSI (1 dB resolution)

MECHANICAL DRAWINGS



BLEPARK

Application protocol specification

V3.1

Changelog

- Changelog for V2.1
  - First release: w.r.t to JunglePass board, the following have been added:
    - keepCtrlAlive() and keepHostAlive() messages to allow both the
- Changelog for V2.2
  - For APP developers only
    - Added routines to calculate MAC to PASSWORD transformation (for APP developers only)
    - Added map of GATT server characteristics
  - Added a NOTE on calibration
  - Added sequence diagram for keep alive messages
  - Added description of keepCtrlAlive() message missing in the previous release
  - Compressed number of ticket types to 2 (transient and subscriptions). Specific types must be addressed in ticket record length.
- Changelog for V3.1
  - Major version updated to allow retro compatibility with hosts attached to Junglepass controllers. The “JunglePass Board V2.X” documents indeed overlap with V2.X of BLEPARK specifications in terms of major protocol number and produces problems in fields.
    - NOTE: every BLEPARK boards shall be interfaced with a major protocol version >= 3. Major version number 2 is reserved for existing installations of Junglepass board.
  - Integrated missing portion related to firmware update protocol subset in the documentation

GATT server characteristic list

```
BLE_UUID_TAS_REQ_ACCESS_CHARACTERISTIC          0x0002
BLE_UUID_TAS_USERID_TICKET_CHARACTERISTIC       0x0003
BLE_UUID_TAS_DATA_TICKET_CHARACTERISTIC         0x0004
BLE_UUID_TAS_TICKET_PROCESSED_CHARACTERISTIC    0x0005
BLE_UUID_TAS_TOGGLE_TICKET_CHARACTERISTIC       0x0006
BLE_UUID_TAS_TICKET_ERROR_CHARACTERISTIC        0x0007
BLE_UUID_TAS_TAC_CHARACTERISTIC                 0x0008
BLE_UUID_TAS_RTX_PWR_CHARACTERISTIC             0x0020
BLE_UUID_TAS_PTX_PWR_CHARACTERISTIC             0x0021
BLE_UUID_TAS_MAC_CHARACTERISTIC                 0x0030
BLE_UUID_TAS_PASSWD_DATA_CHARACTERISTIC         0x0031
BLE_UUID_TAS_PASSWD_CTRL_CHARACTERISTIC         0x0032
BLE_UUID_TAS_HOST_STATUS_CHARACTERISTIC         0x0033
```

Device calibration

NOTE ON CALIBRATION

Calibrator has to write to characteristic 0x0021 to set the transmitted power value. This is done automatically by Operator APP.

Firmware version “blepark_app_V1_2” stills allows to set such value without previous authentication. From blepark_app_V1_3 this is no more possible.

CRC generation for password generation (only for APP team)

```
#define WIDTH  (8 * sizeof(crc))
#define TOPBIT (1 << (WIDTH - 1))
#define POLYNOMIAL 0x04C11DB7

crc crcGen(uint8_t *message, int nBytes)
{
        crc  remainder = 0;

        for (int byte = 0; byte < nBytes; ++byte)
        {
        remainder ^= (message[byte] << (WIDTH - 8));

        for (uint8_t bit = 8; bit > 0; --bit)
        {
        if (remainder & TOPBIT)
        {
        remainder = (remainder << 1) ^ POLYNOMIAL;
        }
        else
        {
        remainder = (remainder << 1);
        }
        }
        }
        return (remainder);
}
```

Message format

The format of messages exchanged between a host (peripheral software) and BLE controller is described in the following table.

Name	Length (bytes)	Description
Module ID	1	Module ID
Opcode	1	Operation Code
Payload Length	2	
Payload	N	

Message format

It’s fields are explained below:

- Module ID: it represents a specific functionality to be addressed. For example, firmware update or bluetooth communication.
- Opcode: identifies the specific message the host or the controller wishes to send to peer.
- Payload length: length of payload
- Payload: message contents

Modules IDs

The following table shows the module ID used in BLEPARK. For re-compatibility with JunglePass protocol, codes do not start from 1.

Modules IDs table	
Short name	**Value (hex)**
HUB_MOD_BLE	0x12
HUB_MOD_NRF_SBL	0x16
HUB_MOD_NONE	0x1B

Opcodes

Host -> BLEPARK

Modules IDs table	
Message name	**Opcode**
sendConfig()	0xaa
reset()	0xac
endTransaction()	0xae
startTransit()	0xf0
endTransit()	0x0f
ticketEmission()	0x81
checkTicketResponse() //same format as ticketEmission()	0x82
keepHostAlive()	0x83

Opcodes

BLEPARK -> Host

Modules IDs table	
Message name	**Opcode**
requestConfig()	0xab
sessionEnd()	0xaf
requestTicket()	0x18
identifyCustomer()	0x19
conf()	0x42
verifyTicket()	0x99
checkTicket() //same format as verifyTicket()	0xa0
keepCtrlAlive()	0xa1

Opcodes

Common between BLEPARK and Host

Modules IDs table	
Message name	**Opcode**
ACK	0xff
NACK	0xf0

Messages

APDU sendConfig(bt,dt,dn,ic)

<8-bit module> HUB_MOD_BLE

<8-bit opcode> 0xaa

<16-bit len> 9+ic_len

<8-bit brand type> (bt) #0

<8-bit device type> (dt) #1

<32-bit device number> (dn), little endian #2-#5

<8-bit major protocol number> #6

<8-bit minor protocol number> #7

<8-bit installation code length> (ic_len) #8

<ic_len-byte installation code> (ic) #9-(#9+#ic_len-1)

NOTE1: the installation code can be either in hex or ascii format.

NOTE2: the maximum length for installation code is 12 bytes

Messages

APDU sendConfig(bt,dt,dn,ic)

dt value	Device type
0x00	Entrance
0x01	Exit
0x04	APM
0x05	Exit+APM

bt value	HUB Brand
0x00	FAAC
0x01	ZEAG
0x02	DATAPARK
0x03	CTR
0x04	HUB

Messages

APDU reset()

<8-bit module> HUB_MOD_BLE

<8-bit opcode> 0xac

<16-bit len> 0

<variable data> null

NOTE: this command logically resets JANUS BLE, i.e. it restarts waiting for start transit event for I/O devices

Messages

APDU startTransit ()

<8-bit module> HUB_MOD_BLE

<8-bit opcode> 0xf0

<16-bit len> 0

<variable data> null

Messages

APDU endTransit(etr)

<8-bit module> HUB_MOD_BLE

<8-bit opcode> 0x0f

<16-bit len> 0x01

<variable data> 8-bit end transit result (etr)

NOTE: see “End Transit Result (etr) codes” section

Messages

End Transit Result (etr) codes

Message	Value
Normal gate crossing	0xff
Gone backward	0xf0
Undertermined	0x18

Messages

APDU ticketEmission (te,tt, gmt_offset, ts, trl, trp, *tte,am,cur*)

<1-byte flag>

<1-byte opcode> 0x81

<2-bytes len> trl + 8 (trl = ticket length) + 11 (only in case of APM and Exit+APM devices)

<1-byte> ticket error (te) #0 //if te != 255 -> other fields must be present (trl can be 0) and they will be ignored.

<1-byte> ticket_type (tt) #1

<1-byte> gmt_offset (int8) - #2 //offset w.r.t to GMT, expressed in quarters of hours (ex.: +5 means +75 minutes from GMT)

<4-bytes> unix timestamp - little endian #3- #6

<1-byte> ticket_record_len (trl) in bytes #7

<trl-bytes> ticket_record_payload (trp) #8...8+trl-1 //SEE “ticket representation“ paragraphs for the specific formatting of each ticket

<4-bytes unsigned int - time to exit> (tte) #trl+8 - #trl+11 //in minutes

<4-bytes float - amount> (am) #trl+12 - #trl+15

<3-bytes - currency> (cur) #trl+16 - #trl+18 //according to ISO 4217 - little endian

NOTE1: See table “Ticket error” for te results.

NOTE2:: tte, am, cur fields are used only in case of APM or Exit+APM devices (code 0x05)

Messages

APDU keepHostAlive()

<8-bit module> HUB_MOD_BLE

<8-bit opcode> 0x83

<16-bit len> 0

...

Messages

APDU keepCtrlAlive()

<8-bit module> HUB_MOD_BLE

<8-bit opcode> 0xa1

<16-bit len> 0

...

Messages

APDU conf (rc)

<8-bit module>

<8-bit opcode> 0x42

<16-bit len> 1

<8-bit> return code (rc) //either APDU_OK or APDU_KO

...

Messages

APDU endTransaction(te,tte,am,cur)

<8-bit module> HUB_MOD_BLE

<8-bit opcode> 0xae

<16-bit len> 12

<8-bit ticket error> (te) #0 //see “Ticket error” table

<32 bit unsigned int - time to exit> (tte) #1 - #4 //in minutes

<32 bit float - amount> (am) #5 - #8

<24 bit - currency> (cur) #9 - #11 //according to ISO 4217 - little endian

Messages

APDU requestConfig()

<8-bit module>

<8-bit opcode> 0xab

<16-bit len> 0x00

<variable-data> NULL

Messages

APDU requestTicket ()

<8-bit module>

<8-bit opcode> 0x18

<16-bit len> 0

Messages

APDU identifyCustomer(id)

<8-bit module>

<8-bit opcode>

<16-bit len> id_len (variable, comprised of separators)

<id_len 8-bit data> id_payload

id_payload format:

CCCC;PNPNPNPNPN;PCPC

Where:

1. CCCC = country code
2. PNPN... = phone number
3. PCPC = pin
4. ; = separator (each field can be variable length)

Messages

APDU sessionEnd(esr)

<8-bit module>

<8-bit opcode> 0xaf

<16-bit len> 0x05

<8-bit> esr //see table “End Session Result (esr)”

Messages

End Session Result (esr)

Decimal Value	Description
0	App updated
1	Timeout
2	BLE problem

Messages

APDU sendStatus()

<8-bit module>

<8-bit opcode> 0xa1

<16-bit len> 0x00

NOTE: currently the pair getStatus/sendStatus are only used to verify the controller is powered-up and running. Host may send a getStatus during inactivity periods with a relaxed period (ex.: 5s)

Messages

APDU verifyTicket(te,tt, gmt_offset, ts, trl, trp)

<1-byte flag>

<1-byte opcode> 0x99

<2-bytes len> trl + 8 (trl = ticket length)

<1-byte> ticket error (te) #0 //if te != 255 -> other fields can be ignored.

<1-byte> ticket_type (tt) #1

<1-byte> gmt_offset (int8) - #2 //offset w.r.t to GMT, expressed in quarters of hours (ex.: +5 means +75 minutes from GMT)

<4-bytes> unix timestamp - little endian #3- #6

<1-byte> ticket_record_len (trl) in bytes #7

<trl-bytes> ticket_record_payload (trp) #8...8+trl-1 //SEE “ticket representation“ paragraphs for the specific formatting of each ticket type

NOTE1: See table “Ticket error (te)” for te results.

Messages

Ticket error (te)

Decimal Value	Description	Generator	Action
Generic validation codes			
255	Success/No error (convenience value)	All	None
254	Undefined error	All	None
253	Ignore field	All	The field must not be used
Errors generated by MS			
0	display standard ***(unused within an app)***		None
1	A difference exists to be paid (excess to be paid)	Exit	Ticket becomes ACTIVE and we need to remember that it was already paid because we have to sum the new received amount with the previous one.
2	already used (exited from parking lot)	APM. Exit	Ticket becomes CONSUMED.

Messages
Ticket error (te) - cont'd

Decimal Value	Description	Generator	Action
3	black list	APM, Exit	Ticket becomes DISPOSED.
4	not paid (in exit to indicate ticket not paid)	Exit	None (shouldn't happen.)
5	wrong gate	Exit	None
6	not valid (for example, ticket not present in MS)	Entrance, APM, Exit	Ticket becomes DISPOSED.
7	read voucher ***(unused within an app, maybe useful with credit cards or similar)***	APM, Exit	None
8	not enough ***(unused within an app)***	APM, Exit	None
9	wrong plate number	Entrance, Exit	None
10	payment canceled in APM	APM	None
11	subscription not valid	Entrance	If app has another subscription available try to use it, otherwise asks the user to take a transient title.

Messages

Ticket error (te) - cont’d

Decimal Value	Description	Generator	Action
12	Parking lot is FULL	Entrance	

Messages

Ticket error (te) - cont'd

Decimal Value	Description	Generator	Action
Errors generated locally (by lane devices or APM)			
100	site code mismatch	APM, Exit	None
101	Ticket format error (ex.: less or more than expected number of characters, bad checksum, device number beyond limit, progressive beyond upper limit)	APM, Exit	None
102	Another transaction is in progress	Entrance, Exit	None
103	**Timeout: actions took too long to be completed (unused within an app, used by peripherals)**	All	None
104	**BLE problem: BLE controller lost connection with central, or some error occurred in the BLE exchange (unused within an app, used by peripherals)**	All	None

Messages

Ticket types

Ticket Types Table		
Code	**Meaning**	**Description**
1	TRANSIENT	Originated by controller
2	SUBSCRIPTION	Originated by APP

Messages

Ticket representation (Ticket Record Payload)

Constraints:

Maximum Ticket Record Length (MTRL): 64 bytes

Messages

Ticket representation (Ticket Record Payload)

UNIQUE FORMAT

<1 byte> identifier length (idl)

<idl - bytes> identifier field (ASCII -> printable on smartphone)

<1 byte> product specific length (psl)

<psl - bytes> product specific binary data

Example:

0x05 0x31 0x32 0x33 0x34 0x35 **0x07** 0x01 0x02 0x03 0x04 0x05 0x06 0x07

Maximum Ticket Record Length (MTRL): 64 bytes

NOTE: the MTRL (maximum ticket record length) comprises the field lengths

Message for firmware update

- The following pages describes the set of commands to update and retrieve information about firmware
- Use the flag value HUB_MOD_NRF_SBL instead of HUB_MOD_BLE when dealing with commands related to firmware.

Messages

SBLopcodes		
Short name	**Value**	Meaning
HUB_SBL_RESP	0x01	Generic response
HUB_SBL_CMD_TRIGGER	0x02	Initiates the fimware upgrade procedure
HUB_SBL_CMD_GET_SBL_VERSION	0x03	Get the bootloader version
HUB_SBL_RESP_GET_SBL_VERSION	0x04	Response to above message
HUB_SBL_CMD_GET_UCODE_VERSION	0x05	Get the User Code version
HUB_SBL_RESP_GET_UCODE_VERSION	0x06	Response to above message
HUB_SBL_ASK_CBLK	0x07	Board asks for a new code block
HUB_SBL_SND_CBLK	0x08	Host sends a code block
HUB_SBL_CMD_FWU_FINISHED	0x09	Board has finished upgrading fw

Messages

SBLgeneric response types		
Short name	**Value**	Meaning
HUB_SBL_RESP_OK	0x01	Previous command was correctly applied
HUB_SBL_RESP_KO	0x02	Error while applying previous commands
HUB_SBL_RESP_WRONG_CRC	0x03	Wrong CRC
HUB_SBL_RESP_UNREC_CMD	0x04	Command not recognized

Messages

HUB_SBL_RESP(resp)

<1-byte flag> HUB_MOD_NRF_SBL

<1-byte opcode>

<2-byte len> 1

<1-byte > resp //see response table

Messages

HUB_SBL_CMD_TRIGGER(resp)

<1-byte flag> HUB_MOD_NRF_SBL

<1-byte opcode>

<2-byte len> 0

Messages

HUB_SBL_CMD_GET_SBL/UCODE_VERSION(resp)

<1-byte flag> HUB_MOD_NRF_SBL

<1-byte opcode>

<2-byte len> 0

Messages

HUB_SBL_CMD_RESP_GET_SBL/UCODE_VERSION(resp)

<1-byte flag> HUB_MOD_NRF_SBL

<1-byte opcode>

<2-byte len> 2

<1-byte> major number

<1-byte> minor number

Messages

HUB_SBL_ASK_CBLK(resp)

<1-byte flag> HUB_MOD_NRF_SBL

<1-byte opcode>

<2-byte len> 4

<4-byte> block_number //each block is 32 byte long

NOTE: in case the *.bin image (ARM format) of the two devices is a multiple of 32 bytes, the device asks for an “out of bound” code block. The host shall answer with a HUB_SBL_SND_CBLK message containing 0 bytes for the code_block section (this means the message payload is composed only of block_num and crc fields.

Messages

HUB_SBL_SND_CBLK(block_num, code_block, crc)

<1-byte flag> HUB_MOD_NRF_SBL

<1-byte opcode>

<2-byte len> 4 (block number) +num_bytes_block + 2 (crc)

<4-byte> block_number //each block is 32 byte long

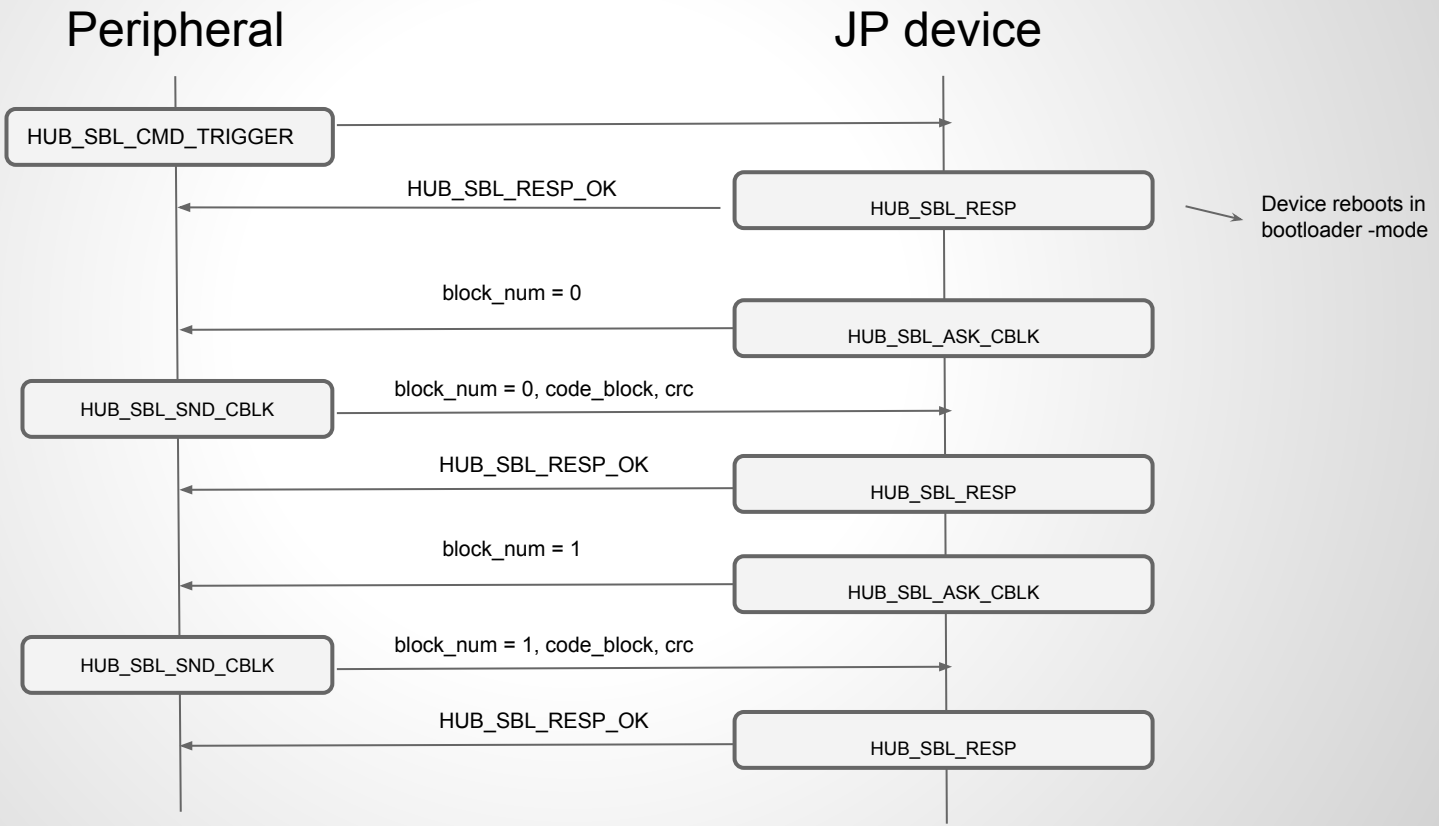
<num_bytes_block-byte> code_block

<2-bytes> crc //calculated taking into account the payload contents (no flag, opcode, length)

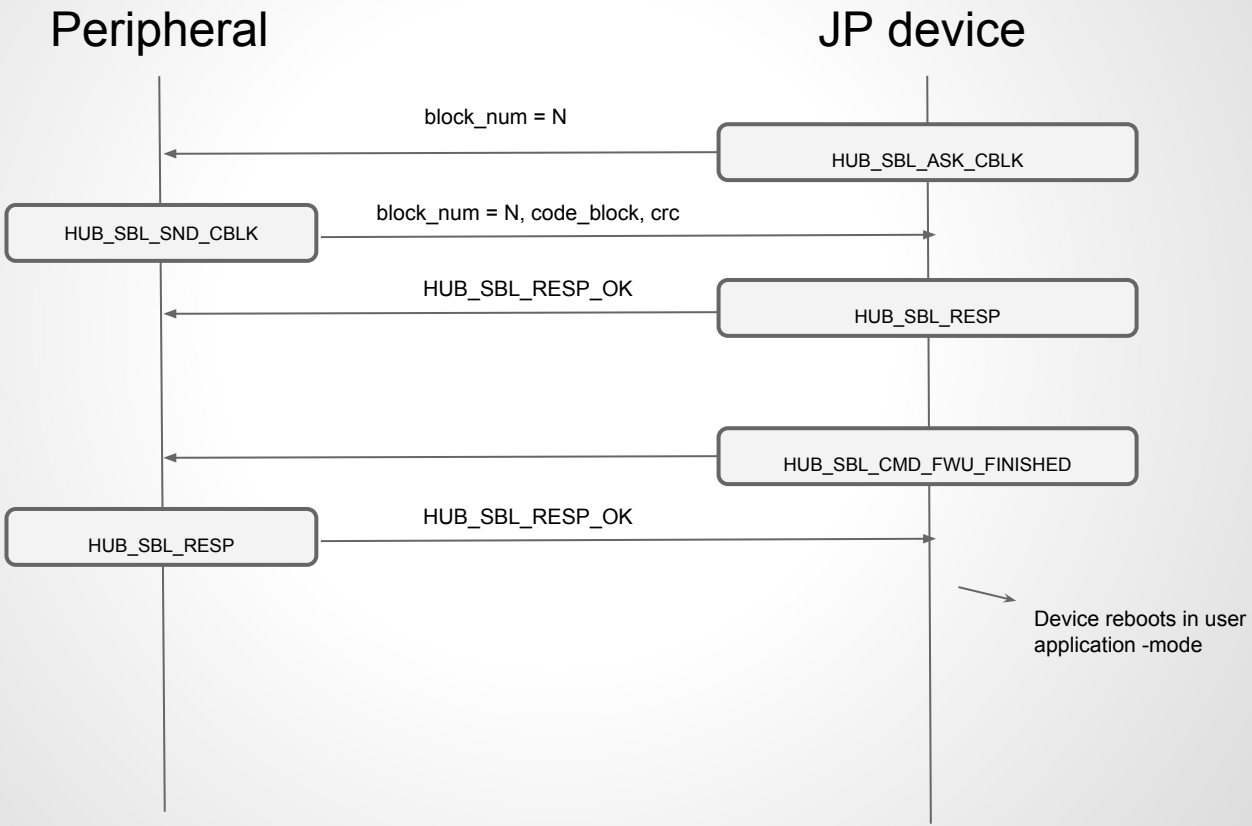
CRC generation

```
ushort table[256] = {0,};
static ushort getCrc16Ccitt(char* bytes, int len)
{
    const ushort poly = 0x08408;
    ushort initialValue = 0x0000;
    ushort temp, a;
    ushort crc = initialValue;
    int i = 0;
    for (i = 0; i < sizeof(table); ++i)
    {
            temp = 0;
            a = (ushort)(i << 8);
            int j = 0;
            for (j = 0; j < 8; ++j)
            {
                    if (((temp ^ a) & 0x8000) != 0)
                            temp = (ushort)((temp << 1) ^ poly);
                    else
                            temp <<= 1;
                    a <<= 1;
            }
            table[i] = temp;
    }
    i = 0;
    for (i = 0; i < len; ++i)
    {
            crc = (ushort)((crc << 8) ^ table[((crc >> 8) ^ (0xff & bytes[i]))]);
    }
    return crc;
}
```

Example fw update - 1/2



Example fw update - 2/2



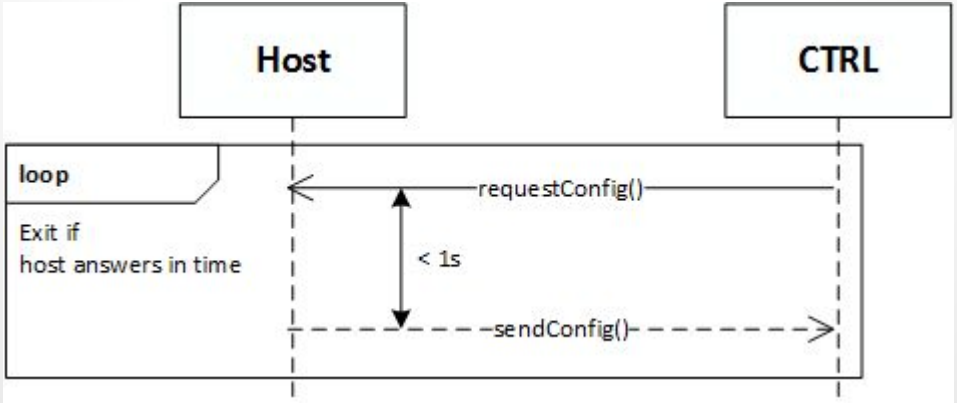
Procedures

USE CASE
Unconfigured controller

Procedures
Description & Sequence diagram

Description:

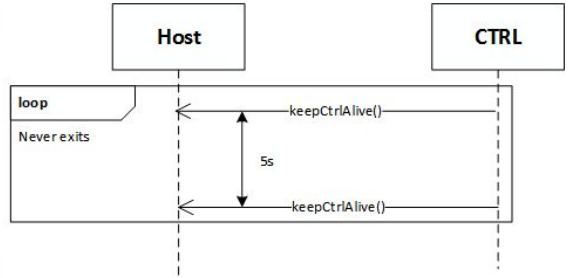
After power-up the controller is not configured. It does not emit any advertising signal until a configuration is sent by host. Configuration is request once every 1s period, through the requestConfig() command. The use case terminates when the host transmit a sendConfig() message with appropriate fields.



Procedures

USE CASE
Keep alive

Procedures
Sequence diagram



Procedures
Description

The host is capable to understand if a device is operative by sampling the keepCtrlAlive() message. It is sent every 5s.

The controller can understand if the Host is ready by sampling its keepHostAlive() message. Such message is emitted by host every 5s.

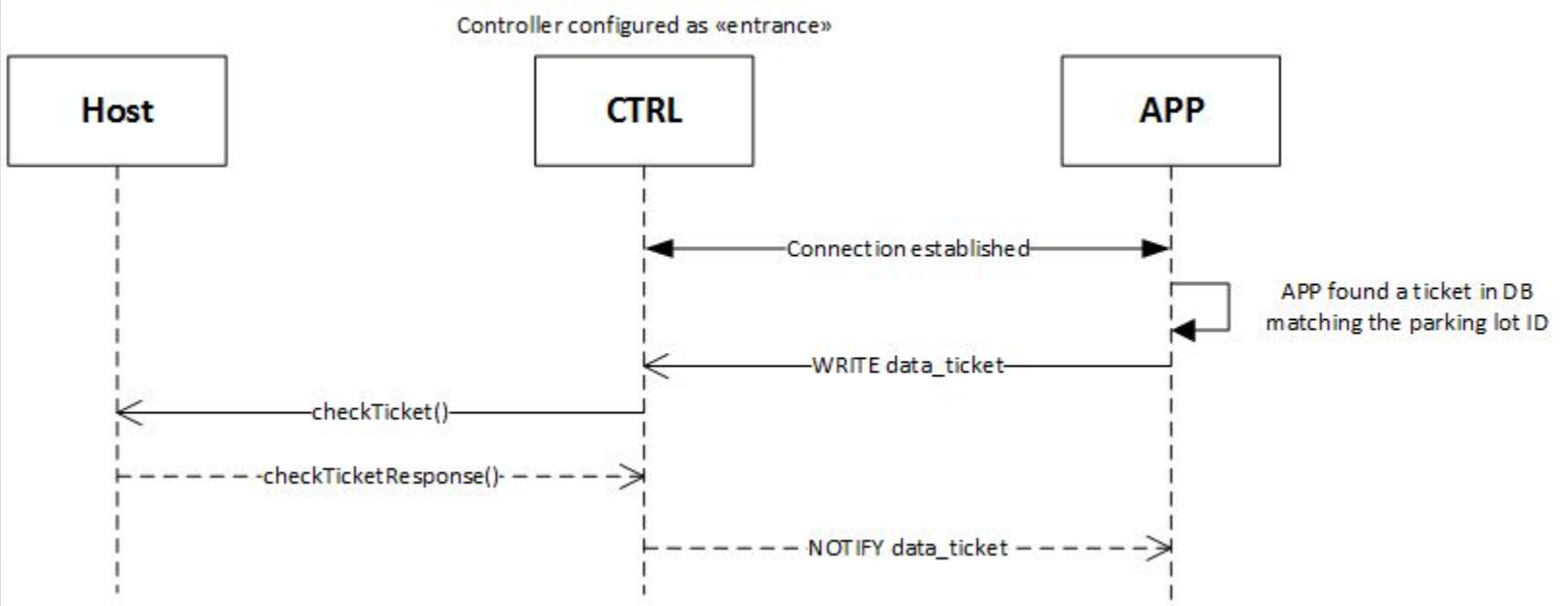
If these signals are not detected for 15 seconds, then

- The host can assume the controller is not ready (removed, or having some problems)
- The controller can assume the host is no more ready (offline)

Procedures

USE CASE
Ticket checking

Procedures
Sequence Diagram



Procedures
Description

The check ticket functionality is used to let the app understand if a specific ticket is valid or not for the system. This can be useful, as explained in the next scenario.

ENTRY SCENARIO

User entered a parking lot with a ticket (transient or subscription). The result of “endTransit” message was not fed to app due to a disconnection, so the app does not know if user moved back or a normal gate crossing was performed. The ticket is consequently still valid in app logic. So we have two scenarios:

1. Normal gate crossing
2. Move back

Irrespective of what happened, user can now engage an entrance device again. Why?

- Because user accidentally engages to an entrance device after a normal gate crossing.
    - The ticket is valid for the system. The app shall ignore the attempt of the user to access the parking again and close connection.
- Because user moved back and then tries to access the parking lot again.
    - The ticket is not valid for the system. The app shall remove the current ticket and request a new ticket to the parking.

So, how the app can know what to do? Use validity check functionality (read later on)!

Procedures
Description

EXIT SCENARIO

User exited a parking lot with a ticket (transient or subscription). The result of “endTransit” message was not fed to app due to a disconnection, so the app does not know if user moved back or a normal gate crossing was performed. The ticket is consequently still valid in app logic. So we have two scenarios:

1. Normal gate crossing
2. Move back

Irrespective of what happened, user can now engage an entrance device again. Why?

- Because user accidentally engages to an entrance device after a move back.
    - The ticket is valid for the system. The app shall ignore the attempt of the user to access the parking again and close connection.
- Because user performed a normal gate crossing and then tries to access the parking lot again
    - The ticket is not valid for the system. The app shall remove the current ticket and request a new ticket to the parking.

So, how the app can know what to do? Use ticket validity functionality!

Procedures
Description

The checkTicketResponse() has the same format of the ticketEmission() message. The most important field is “ticket error”. It hold two values in this message:

- 0xff (no error): this means the ticket is valid for the parking system -> then app keeps the ticket in DB and closes the current connection.
- 0x06 (ticket not valid): the app removes the ticket and proceeds with other use cases, keeping the connection active

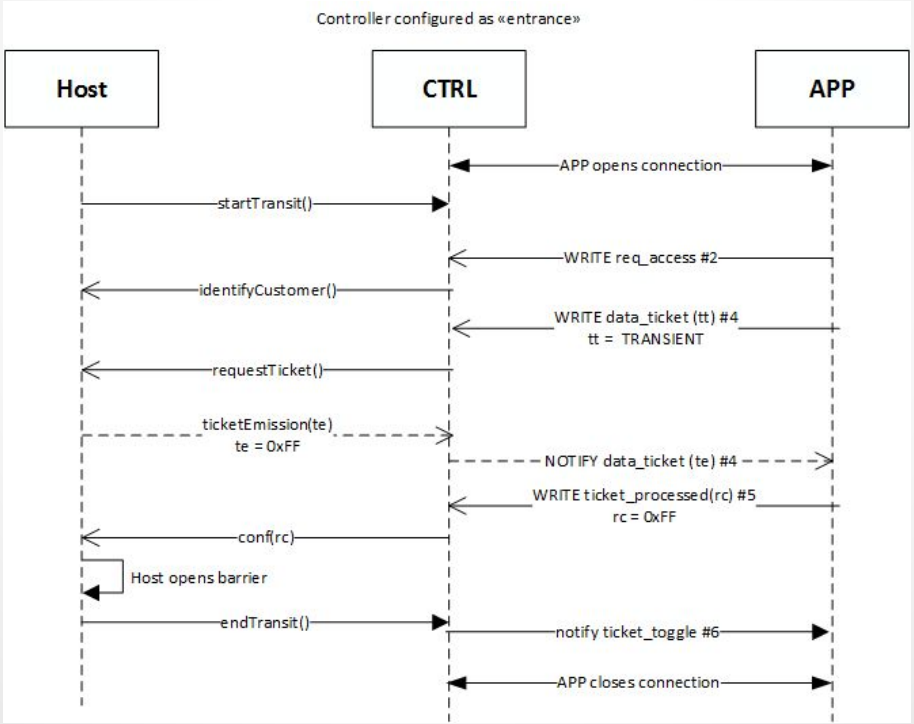
Additional note:

- The ticket checking functionality is required only for entrance devices

Procedures

USE CASE
Entrance

Procedures
Transient - happy path



Procedures
Description

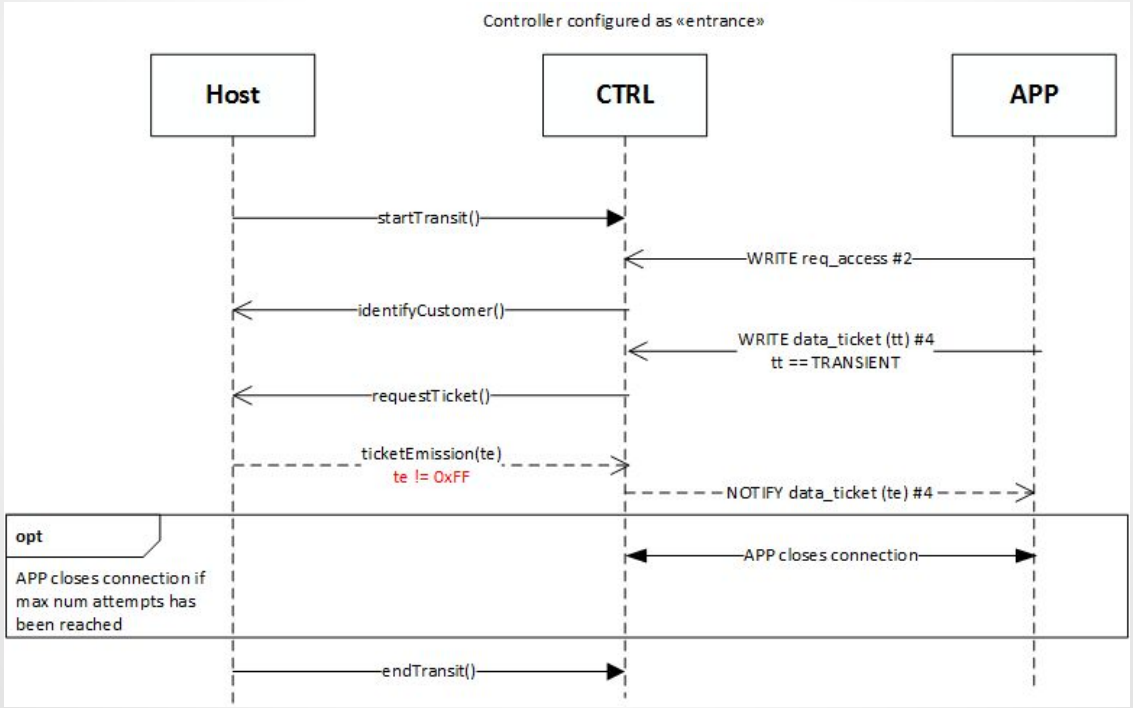
NOTES

- Connection establishment and loop engagement are unrelated. The condition for the controller to send a requestTicket() (or verifyTicket()) is that BOTH event occurred.
- A consequence of this is that identifyCustomer() and startTransit() can occur in any order.
    - This is true not only for entrance but also for exit use case.
    - This is true irrespective of the ticket type chosen

- The ticketEmission() command is used in two ways:
    - in the “classic” sense in case of transient ticket for entrance devices.
    - But It is also used in exit and payment machines: in these cases the host must:
        - Update the ticket record if needed, for offline management
        - Always update all the the auxiliary fields (timestamp…) to the most recent values. Please remember: fields outlined in command descriptions are totally unrelated to ticket internal contents. The controller is not aware of internal ticket meaning of both visible and binary ticket portions.

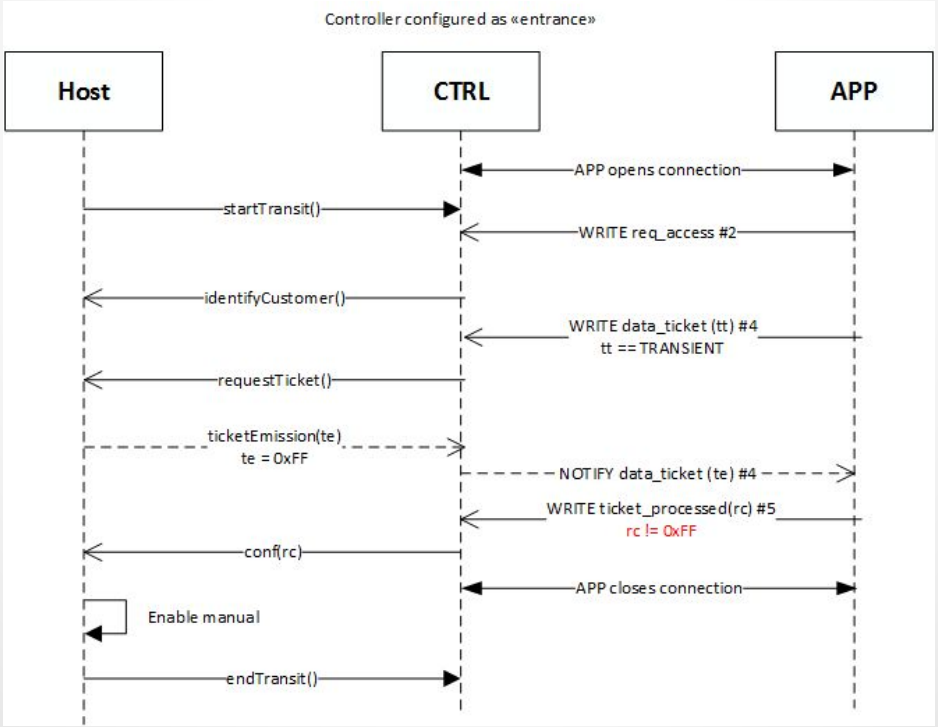
Procedures

Transient - host cannot dispense a ticket

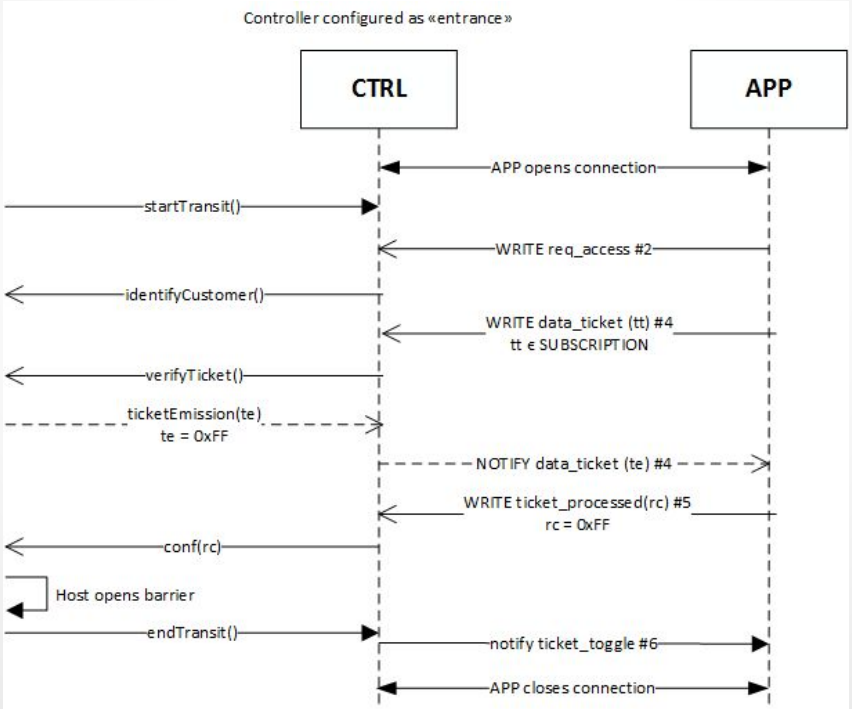


Procedures

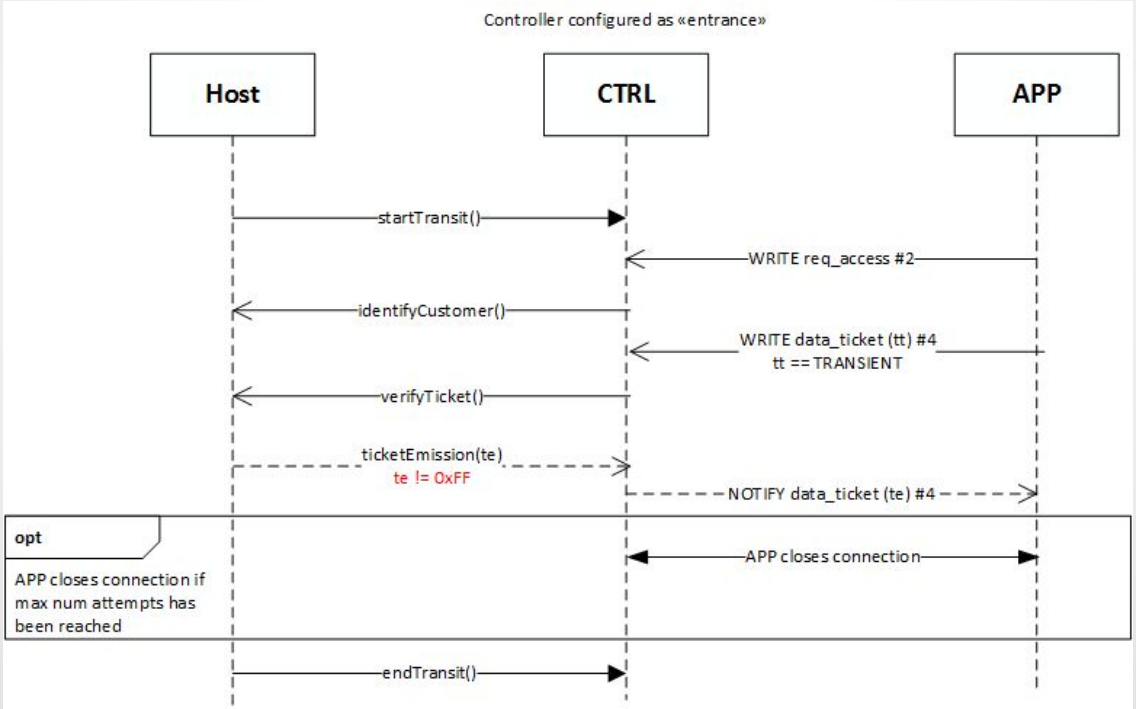
Transient - app cannot process ticket



Procedures
Subscription - happy path

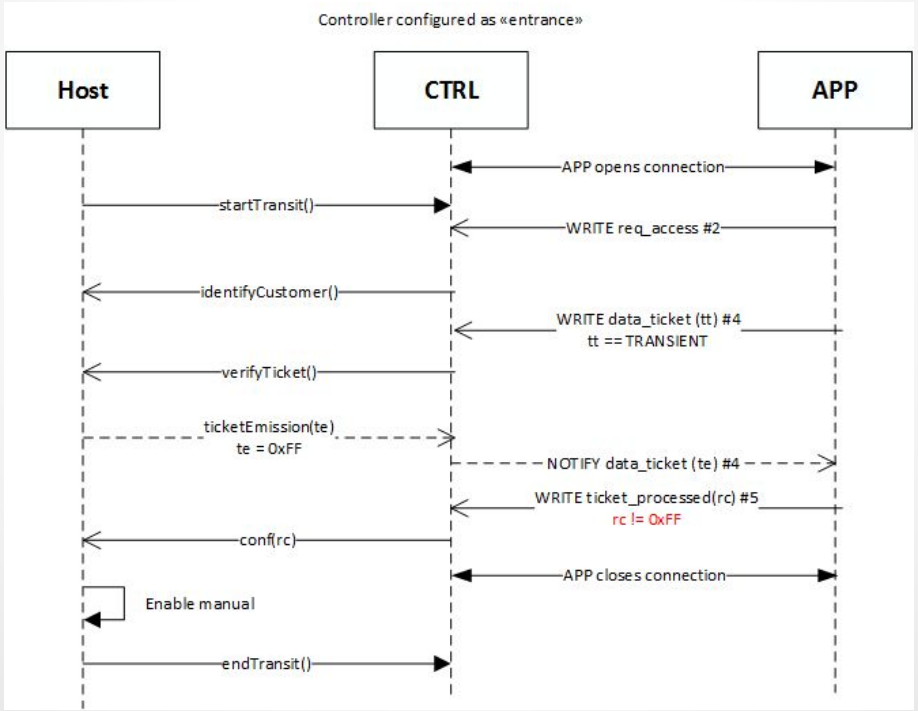


Procedures
Subscription - verification error



Procedures

Subscription - app cannot process ticket

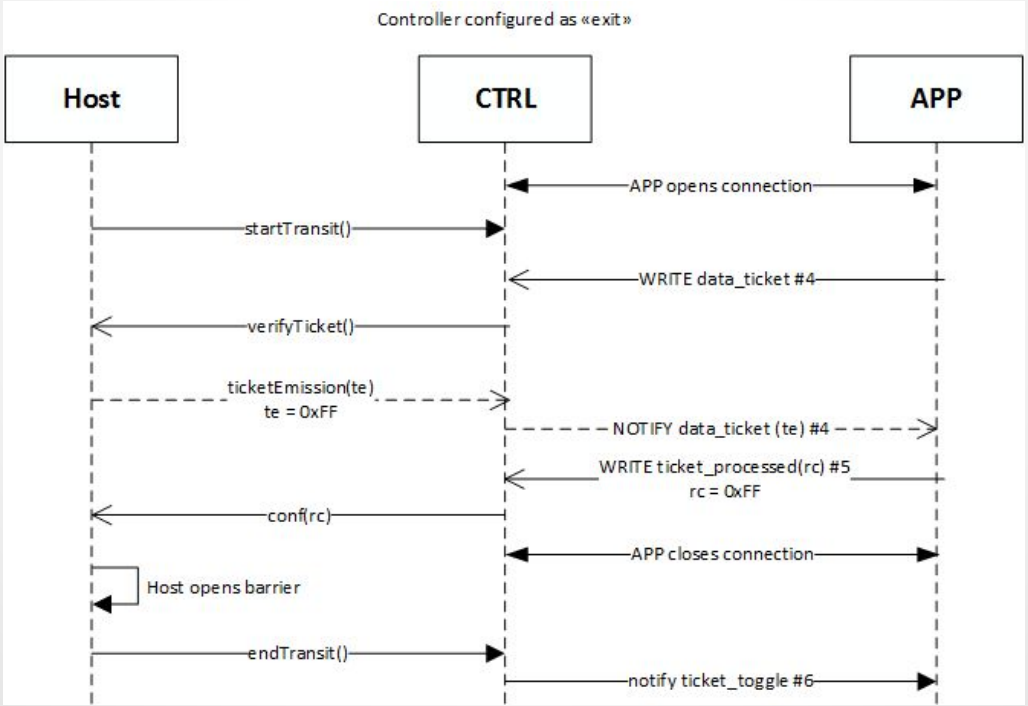


Procedures

USE CASE
Exit

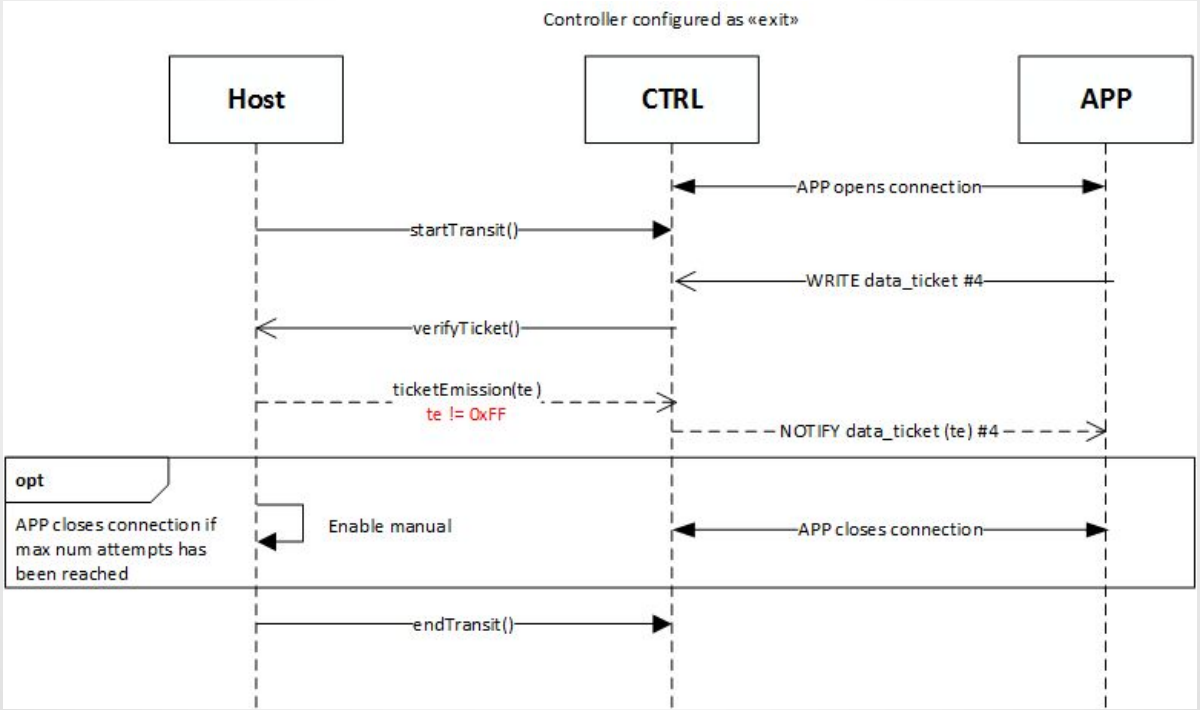
Procedures

Transient & Subscription - happy path



Procedures

Transient & Subscription - verification negative



Procedures

USE CASE
Payment

Procedures
Pure BLE-based payment