CMBS HAN SERVER PROTOCOL

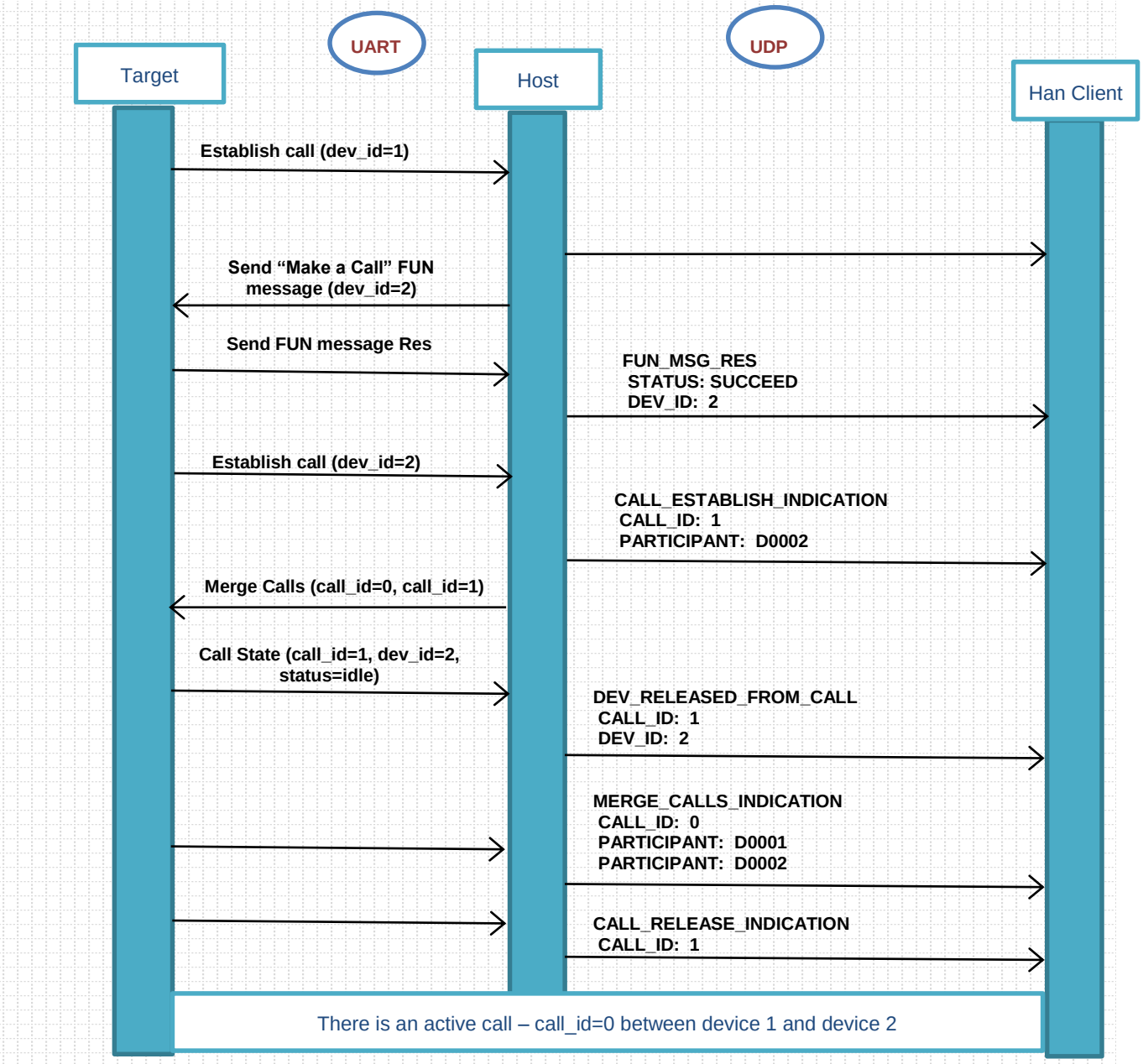
(Rev 1.18)

Revision History

Revision	Date	Author	Description
0.1	24 July 2014	Avraham Fraenkel	Initial creation.
0.3	30 July 2014	Avraham Fraenkel	Improvements after feedback.
0.4	6 Aug 2014	Avraham Fraenkel	Init Response; add EMC parameter to Table; registration flow.
0.5	25 Aug 2014	Avraham Fraenkel	RFPI added to INIT_RES; corrections of documentation, add UNIT_ID to Device Table Response.
0.6	11 Sep 2014	Avraham Fraenkel	Additional FUN "DATA:" format was changed to hex (both ways), BASE_RFPI was deleted from INIT response
0.7	9 Feb 2015	Avraham Fraenkel	Add DEV_INDEX parameter to DEV_TABLE Support RELEASE_LINK of a device Protocol Changes after version 0.6 are signed RED
0.8	27 Sep 2016	Tamir Aviv	Added control messages: KEEP_ALIVE, KEEP_ALIVE_RES, ERROR, TARGET
0.9	6 Oct 2016	Moria Aharon	Added option to get target state response. Edited TARGET_STATE event
1.0	16 Nov 2016	Moria Aharon	Added new commands description
1.1	23 Nov 2016	Moria Aharon	Added SET_RF_STATE command
1.2	19 Dec 2106	Moria Aharon	Added new commands for getting HW and SW versions
1.3	16 Jan 2017	Moria Aharon	Added new command for resetting target
1.4	29 Jan 2017	Ido Achrak	Rework document
1.5	12 Feb 2017	Moria Aharon	Added new commands for clearing FUN msg Q and getting the number of messages in Q
1.6	12 Jun 2017	Moria Aharon	Added FUN message response Added HS number indication upon HS subscription
1.7	03 Oct 2017	Moria Aharon	Added changes to DEV_TABLE and DEV_INFO responses

1.8	08 Nov 2017	Moria Aharon	Added new command + response: GET_DEV_TABLE_PHASE_2, DEV_TABLE_PHASE_2
1.9	14 Nov 2017	Omri Lindenberg	Added new command + response: GET_DEV_INFO_PHASE_2, DEV_INFO_PHASE_2
1.10	15 Nov 2017	Moria Aharon	Added EEPROM parameter list in appendix A
1.11	11 Dec 2017	Moria Aharon	Added support for deleting device locally
1.12	02 Jan 2018	Moria Aharon	Added new command + response: BLACK_LIST_DEV_TABLE BLACK_LIST_DEV_TABLE_RES
1.13	15 May 2018	Anjana Hari	Added new command + response: GET_MAX_NUM_OF_DEVICES MAX_NUM_OF_DEVICES_GET_RES GET_NUM_OF_REG_DEVICES GET_NUM_OF_REG_DEVICES_RES Edited typo error for "Clear Messages in FUN MSG Q" section header
1.14	15 May 2018	Moria Aharon	Added Dual Mode commands and indications
1.15	26 June 2018	Anjana Hari	Edited response for Get Black List Device table syntax to BLACK_LIST_DEV_TABLE
1.16	13 Aug 2018	Anjana Hari	Added new command and response: GET_ACTIVE_CALLS_INFO GET_ACTIVE_CALLS_INFO_RES
1.17	28 Nov 2018	Sandeep Sukumar	Added new commands for ULE data call.
1.18	14 Dec 2018	Sheetal Noronha	Added command responses for NEMo and Firmware Upgrade Start and End. (Section 4.3.25 to 4.3.29)

Table Of Contents



1. Introduction

1.1 Purpose

The purpose of this document is to describe CMBS API for HAN server, i.e. the protocol for an external application to control the HAN capabilities of CMBS.

1.2 Scope

1.3 Definitions, Acronyms and Abbreviations

Name	Description
CMBS	Cordless module base station
HAN	Home Automation
HS	Handset
FUN	Functional ULE network
FP	Fixed part of DECT
Device	Any HAN device or Portable part that is connected to FP over DECT

1.4 References and Bibliography

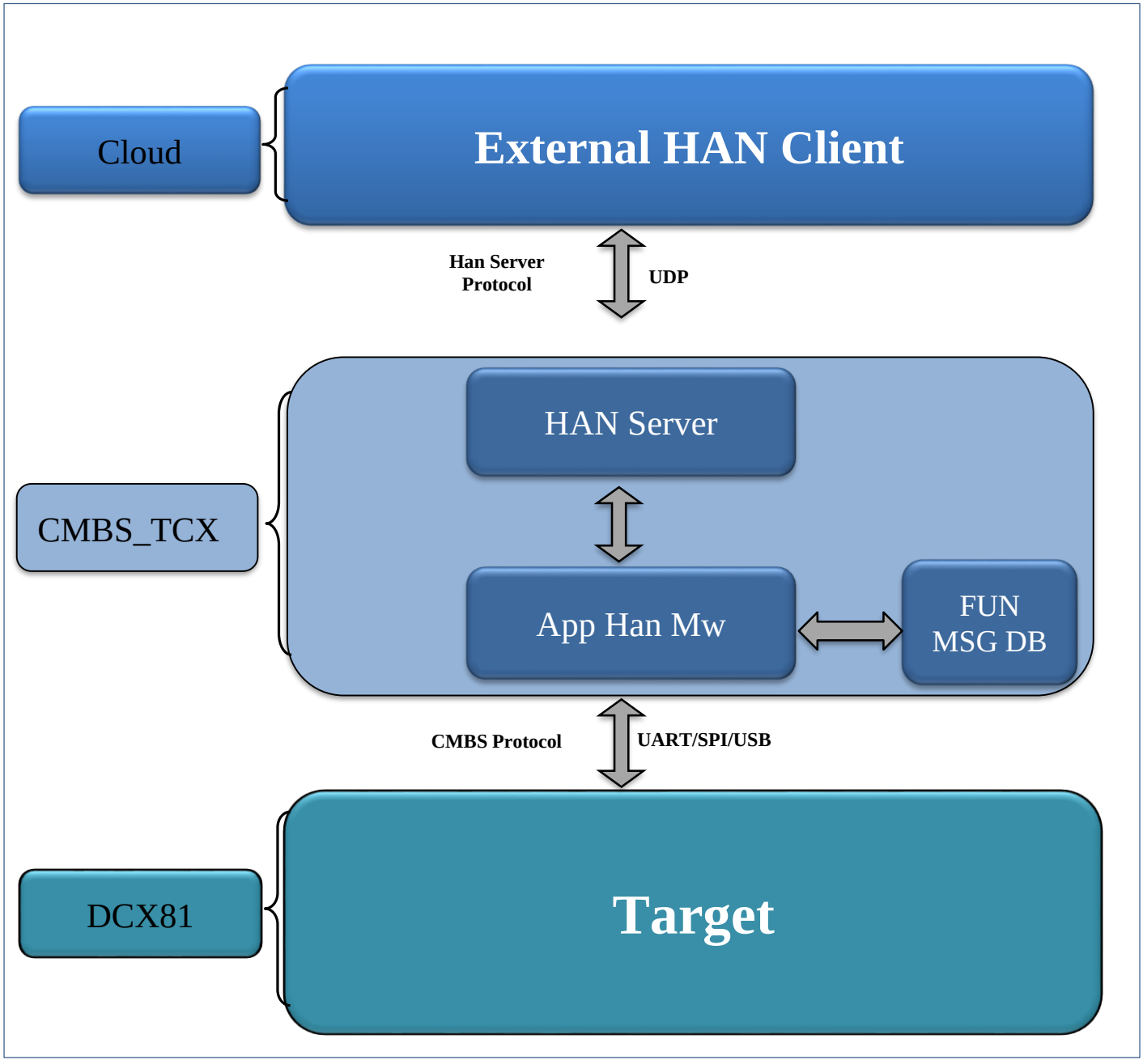
#	Document Name	Version	Date	Location
[1]	FUN standard	1.3	04/07/2016	
[2]	CMBS-HAN-API	2.8	17/01/2016	
[3]				
[4]				

2. CMBS API for HAN server - overview

2.1 Introduction

CMBS API for HAN server divided into Control, FUN, service and debug messages.

The HAN server entity is running under the CMBS host application context and communicates with external HAN clients over UDP.



2.2 CMBS API messages

Control messages:

INIT	App -> CMBS	Notification for Init session with CMBS HAN server
INIT_RES	CMBS -> App	Response Notification for Init
OPEN_REG	App -> CMBS	Ask CMBS to open DECT Registration window
OPEN_RES	CMBS -> App	Response for OPEN_REG request
CLOSE_REG	App -> CMBS	Ask CMBS to close DECT Registration window
CLOSE_RES	CMBS -> App	Response for CLOSE_REG request
REG_CLOSED	CMBS -> App	Registration closed (with Reason)
GET_DEV_TABLE	App -> CMBS	Ask CMBS to send device table (one device or all).
DEV_TABLE	CMBS -> App	Chunk of Devices from device table with their details
GET_DEV_INFO	App -> CMBS	Single Device details (by its DECT ID)
DEV_INFO	CMBS -> App	Single Device details
DELETE_DEV	App -> CMBS	Request to delete a device
DEV_REGISTERED	CMBS -> App	Unsolicited notice about registered device
DEV_DELETED	CMBS -> App	Unsolicited notice about deleted device
KEEP_ALIVE	CMBS -> App	Ask App for keep alive indication
KEEP_ALIVE_RES	App -> CMBS	Response for KEEP_ALIVE request
TARGET_STATE	CMBS -> App	Unsolicited notice about Target status
ERROR	CMBS -> App	Notification about errors
RELEASE_LINK	App -> CMBS	Release link of a device
GET_DEV_TABLE_PHASE_2	App -> CMBS	Ask CMBS to send device table (one device or all) according to phase 2 structure
DEV_TABLE_PHASE_2	CMBS -> App	Chunk of Devices from device table with their details according to phase 2 structure
GET_DEV_INFO_PHASE_2	App -> CMBS	Single Device details (by its DECT ID) according to phase 2 structure
DEV_INFO_PHASE_2	CMBS -> App	Single Device details according to phase 2 structure
BLACK_LIST_DEV_TABLE	App -> CMBS	Ask CMBS to send black list device table

FUN messages:

FUN_MSG	CMBS <--> App	Send or receive a FUM message to / from a device
FUN_MSG_RES	CMBS -> App	Send FUN message response

Service Messages:

GET_EEPROM_BYTES	App -> CMBS	Read Bytes from Target EEPROM request (flex)
GET_EEPROM_BYTES_RES	CMBS -> App	Read Bytes from Target EEPROM response
SET_EEPROM_BYTES	App -> CMBS	Write Bytes from Target EEPROM request (flex)
SET_EEPROM_BYTES_RES	CMBS -> App	Write Bytes from Target EEPROM response
GET_EEPROM_PARAM	App -> CMBS	CMBS Parameter read request (logical)
GET_EEPROM_PARAM_RES	CMBS -> App	CMBS Parameter read response
SET_EEPROM_PARAM	App -> CMBS	CMBS Parameter write request (logical)
SET_EEPROM_PARAM_RES	CMBS -> App	CMBS Parameter write response
GET_PRODUCTION_PARAM	App -> CMBS	CMBS Production parameter read request (logical)
GET_PRODUCTION_PARAM _RES	CMBS -> App	CMBS Production parameter read response
SET_PRODUCTION_PARAM	App -> CMBS	CMBS Production parameter write request (logical)
SET_PRODUCTION_PARAM _RES	CMBS -> App	CMBS Production parameter write response
GET_EEPROM_SIZE	App -> CMBS	Get Target EEPROM size request

GET_EEPROM_SIZE_RES	CMBS -> App	Get Target EEPROM size response
SET_RF_STATE	App -> CMBS	Configure RF state
GET_TARGET_HW_VERSION	App -> CMBS	Get Target HW version
GET_TARGET_HW_VERSION _RES	CMBS -> App	Get Target HW version response
GET_SW_VERSION	App -> CMBS	Get SW version
GET_SW_VERSION_RES	CMBS -> App	Get SW version response
RESET_TARGET	App -> CMBS	Target reset request
GET_NUM_OF_FUN_MSG_IN_Q	App -> CMBS	Get number of messages in FUN MSG Q
GET_NUM_OF_FUN_MSG_IN_Q_RES	CMBS -> App	Get number of messages in FUN MSG Q response
CLEAR_FUN_MSG_Q	App -> CMBS	Clear messages in FUN MSG Q
GET_NUM_OF_REG_DEVICES	App->CMBS	Get number of registered devices
GET_NUM_OF_REG_DEVICES_RES	CMBS->App	Get number of registered devices response
GET_MAX_NUM_OF_DEVICES	App->CMBS	Get maximum number of devices supported
GET_MAX_NUM_OF_DEVICES_RES	CMBS->App	Get maximum number of devices supported response

Debug Messages:

GET_RAM_BYTES	App -> CMBS	Read RAM bytes request
GET_RAM_BYTES_RES	CMBS -> App	Read RAM bytes response
SET_RAM_BYTES	App -> CMBS	Write RAM bytes request
SET_RAM_BYTES_RES	CMBS -> App	Write RAM bytes response

Call Messages:

CALL_ESTABLISH_INDICATION	CMBS -> App	Indication for call established
DEV_RELEASED_FROM_CALL	CMBS -> App	Indication for device released from call
HS_RELEASED_FROM_CALL	CMBS -> App	Indication for HS released from call
MERGE_CALLS_INDICATION	CMBS -> App	Indication for call merge
CALL_RELEASE_INDICATION	CMBS -> App	Indication for call released
CALL_RELEASE	App -> CMBS	Release call request

2.3 CMBS Configuration

2.3.1 Read Device Table

Read HAN device table by chunks due to fragmentation in the CMBS target, HAN application can read the table from DEV_INDEX 0 by chunks of 5 devices each time, till the response is less than 5 devices. Then application knows that the table is fully read.

2.3.2 Number of supported devices configuration

The HAN server MAX_ULE_DEVICE_HOST should be verified and set to the number of ULE devices that the project support + 1 (0 -> 1 indexing), this can be checked in apphan.c file.

3. General Aspects of Protocol

3.1 HAN Server various routes

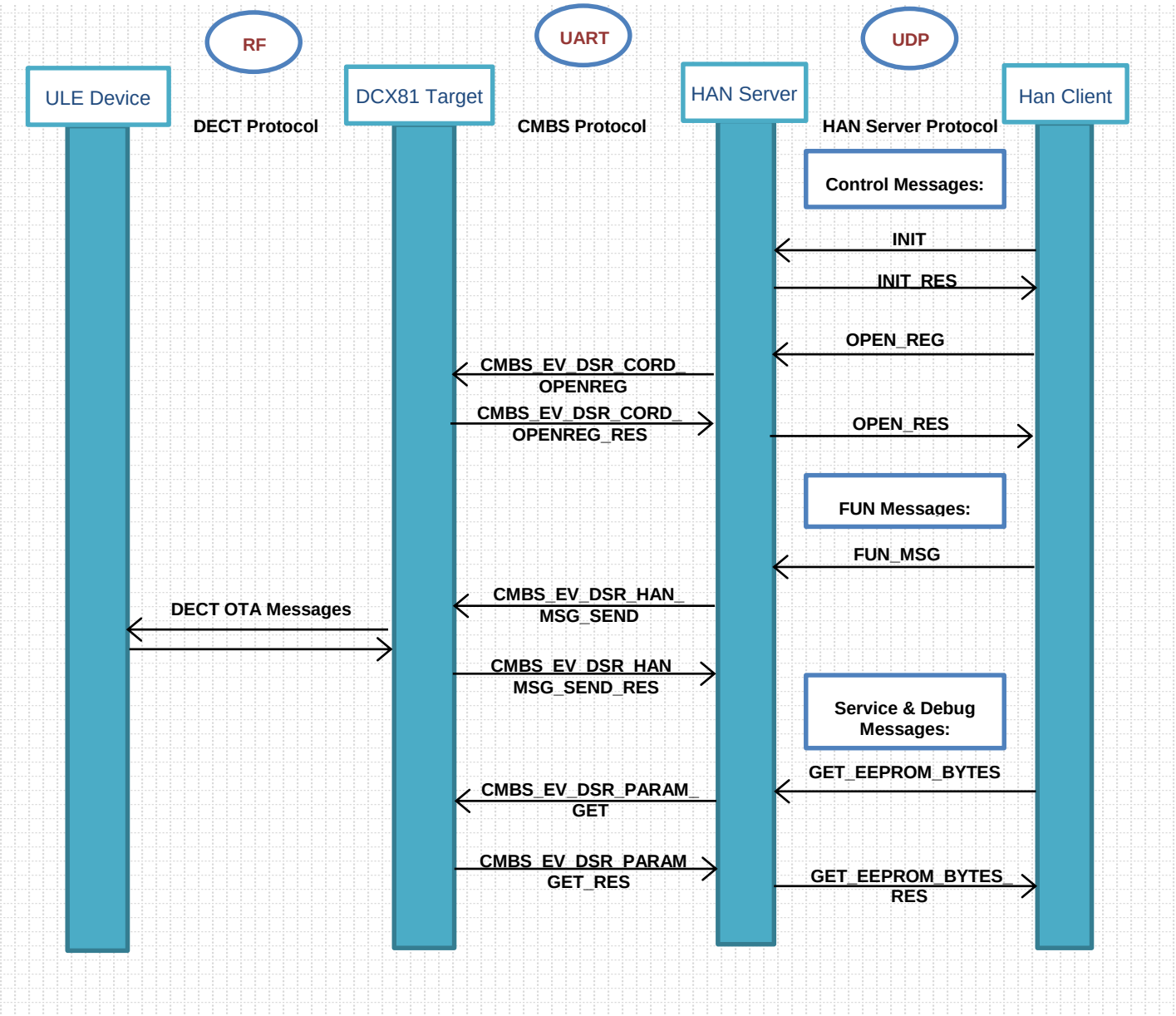


Figure 1: HAN server Messages Types

- HAN-Server control messages flow between the HAN client and the CMBS HAN server. HAN Server can handle locally the received control messages (e.g. INIT) or communicates with CMBS FP (over CMBS protocol) to retrieve the require data.
- HAN server control/service/debug messages do not reach the HAN device.
- HAN Server FUN messages are sent to the HAN devices and received from the HAN devices OTA.

3.2 Transport

Communication between HAN application and HAN server is by a UDP socket.

HAN server is listening on port 3490.

Application should connect to socket and should send an INIT command.

3.3 General Message Format

3.3.1 Message Structure

The protocol is ASCII based.

Each command is built from a few consecutive lines. Each line ends with \r\n.

First line contains Service Name between [], for example: [SRV], [DBG].

Second line contains the Command Name.

Next lines are the parameters, if any, each parameter by a separate line.

Each message ends with 2 blank lines (\r\n\r\n).

3.3.2 Service Name

Each message has a single Service name.

Service name is one word.

It starts at the beginning of first line.

For backward compatibility, messages which are described in HAN SERVER Protocol version 0.9 are also supported without service name.

3.3.3 Command Name

Each message has a single command name.

Command name is one word.

It starts at the beginning of first line.

3.3.4 Parameters

Each parameter is located in a separate line.

Parameters start with a single blank after the LF (“\r\n”).

Generally a parameter format is:

NAME: value

Parameter NAME is capital letters.

Values of UNIT_TYPE and INTRF_IDs (and other FUN message parameters) are according to FUN standard documents.

Unless otherwise specified, values are decimal numbers of the parameter. Generally values are byte or words (according to parameter type).

If a sequence of values is needed (generally in data Payload), they are sent byte by byte with a blank as the delimiter. Generally this will be preceded by a special parameter LEN.

Endian Payload bytes: MSB to LSB (as in FUN standard)

4. Specific HAN Server Messages syntax

4.1 Control Messages

4.1.1 INIT

Syntax:

INIT

VERSION: X

Parameters:

VERSION: protocol version number of application (e.g. 1)

4.1.2 INIT Response

Syntax:

INIT_RES

VERSION: X

~~BASE_RFPI: 0 253 84 8 1~~ (Implementation was postponed for now)

Parameters:

VERSION: protocol version number of HAN Web Server(e.g. 1)

BASE_RFPI: Identification of the HAN server, the DECT base RFPI.

4.1.3 Open Registration Request

Syntax:

OPEN_REG

TIME: X (seconds)

Parameters:

TIME: Open registration for X seconds.

4.1.4 Open Registration Response

Syntax:

OPEN_RES

SUCCEED / FAIL

Parameters:

SUCCEED or FAIL

4.1.5 Close Registration Request

Syntax:

CLOSE_REG

Parameters:

None.

4.1.6 Close Registration Response

Syntax:

CLOSE_RES

SUCCEED / FAIL

Parameters:

SUCCEED or FAIL

4.1.7 Registration Closed

Syntax:

REG_CLOSED

REASON: TIMEOUT / DEV_REGISTERED / HS_REGISTERED

HS_NUMBER: x (only when reason is HS registered)

Parameters:

REASON: the reason for the closed registration window

HS_NUMBER: HS number of the registered HS

4.1.8 Get Device Table Request

General: Read HAN device table by chunks. DEV INDEX is the index in the table (not the DEV DECT ID).

Application can read the table from DEV_INDEX 0 by chunks of 5 devices each time, till the response is less than 5 devices. Then application knows that the table is fully read.

Syntax:

GET_DEV_TABLE

DEV_INDEX: X

HOW_MANY: Y

Parameters:

DEV_INDEX – An index (of HAN table) of the first required device (start with 0)

HOW_MANY – How many devices to return

4.1.9 Device Table Response

The response sends number of devices in this response. Each device has its DECT ID, IPUI, and number of UNITS. For each UNIT, it has its type and number of INTERFACEs. For each INTERFACE, it has a TYPE (server or client) and an ID.

Note: Mandatory UNITSs (as UNIT 0) and mandatory INTERFACEs (for each UNIT type) are not listed.

Note: The indentation below is for illustration needs only. All lines should start with a single blank only.

Syntax:

DEV_TABLE
DEV_INDEX: 0
NO_OF_DEVICES: 2
DEV_ID: 2
 DEV_IPUI: 0 254 85 8 0
 DEV_EMC: 250 16
 NO_UNITS: 1
 UNIT_ID: 3
 UNIT_TYPE: 7
  NO_OF_INTRF: 1
  INTRF_TYPE: 0 / 1 (server / client)
  INTRF_ID: 256
DEV_ID:5
 DEV_IPUI: 0 253 84 8 1
 DEV_EMC: 250 16
 NO_UNITS: 2
 UNIT_ID: 3
 UNIT_TYPE: X
  NO_OF_INTRF: 3
  INTRF_TYPE: 0 / 1 (server / client)
  INTRF_ID: 256
  INTRF_TYPE: 0 / 1 (server / client)
  INTRF_ID: 257
  INTRF_TYPE: 0 / 1 (server / client)
  INTRF_ID: 258
 UNIT_ID: 4
 UNIT_TYPE: Y
  NO_OF_INTRF: 2
  INTRF_TYPE: 0 / 1 (server / client)
  INTRF_ID: 256
  INTRF_TYPE: 0 / 1 (server / client)
  INTRF_ID: 257

Etc.

Parameters:

DEV_INDEX – index (in HAN table) of first device of this message (start with 0).

NO_OF_DEVICES – Number of Devices in this message (currently not including handsets)

Now comes the list of the devices.

For each device:

DEV_ID – unique DECT device ID

DEV_IPUI – IPUI (DECT mac) of the device (5 bytes)

DEV_EMC – EMC (DECT manufactory code) of the device (2 bytes)

NO_UNITS – how many non-mandatory units the device supports (generally manager unit + 1).

UNIT_ID – the ID of this UNIT in that device

UNIT_TYPE – unit type as defined in FUN standard.

NO_OF_INTRF – Number of non mandatory INTERFACES in this UNIT.

INTRF_TYPE -- If server (0) or Client (1)

INTRF_ID – the proprietary INTERFACE ID.

<Possible Additional interfaces for this unit>

<Possible Additional units for this device>

4.1.10 Get Device Info Request

Get specific DEVICE details (from table) by its DECT ID.

Syntax:

GET_DEV_INFO

DEV_ID: 5

Parameters:

DEV_ID – the device DECT ID to be investigated

4.1.11 Device Info Response

Generally, this message contains same info as in Device Table Response, for a single specific device only.

Syntax:

DEV_INFO

DEV_ID: 5

DEV_IPUI: 0 253 84 8 1

DEV_EMC: 250 16

NO_UNITS: 2

UNIT_ID: 0

UNIT_TYPE: X

NO_OF_INTRF: 3

INTRF_TYPE: 0 / 1 (server / client)

INTRF_ID: 256

INTRF_TYPE: 0 / 1 (server / client)

INTRF_ID: 257

INTRF_TYPE: 0 / 1 (server / client)

INTRF_ID: 258

UNIT_ID: 1

UNIT_TYPE: Y

NO_OF_INTRF: 2

INTRF_TYPE: 0 / 1 (server / client)

INTRF_ID: 256

INTRF_TYPE: 0 / 1 (server / client)

INTRF_ID: 257

Parameters:

DEV_ID – DECT ID of the device

DEV_IPUI: IPUI of the device

DEV_EMC – EMC (DECT manufactory code) of the device (2 bytes)

ULE_CAPABILITIES – capabilities supported by the device (1 byte):

- 0 – ULE capabilities not supported
- 1 – ULE phase 1 capabilities
- 3 – ULE phase 1.2 capabilities
- 5 – ULE phase 2 capabilities
- 7 – ULE phase 3 capabilities

ULE_PROTOCOL_ID – protocol id implemented by the device (1 byte):

- 0 – Protocol id undefined
- 1 – ULE protocol 1
- 2 – ULE protocol 2

ULE_PROTOCOL_VERSION - protocol version implemented by the device (1 byte):

- 0 - Protocol version undefined
- 1 – protocol version 1.0
- 2 – protocol version 1.4

NO_UNITS – Number of supported Units by the device. Unit 0 is the general management unit.

For each unit:

UNIT_ID – the ID of this UNIT in that device

UNIT_TYPE

NO_OF_INTRF – number of interfaces that the Unit supports

For each Interface:

INTRF_TYPE: 0 / 1 (server / client)

INTRF_ID -- ID of the interface

4.1.12 Delete a device Request

Syntax:

DELETE_DEV

DEV_ID: 5

DEL_TYPE: LOCAL/ BLACK_LIST

Parameters:

DEV_ID – the device DECT ID to be deleted

DEL_TYPE – indicating the deletion method. In case this field is not sent, device will be deleted via black list. For deleting all devices (DEV_ID: 65535) only local deletion is supported.

4.1.13 Device Registered Indication

Syntax:

DEV_REGISTERED

DEV_ID: 2 (Note: Previous version implementation sent here SRC_DEV_ID)

Parameters:

DEV_ID – the ID of the deleted device

4.1.14 Device Deleted Indication

Syntax:

DEV_DELETED

DEV_ID: 5 (Note: Previous version implementation sent here SRC_DEV_ID)

Parameters:

DEV_ID – the ID of the deleted device.
In case of local deletion: this message will be sent once device is removed from EEPROM

In case of network deletion - this message will be sent once device is informed it is deleted

4.1.15 Release Link of a device Request

Syntax:

RELEASE_LINK

DEV_ID: 5

Parameters:

DEV_ID – the device DECT ID its link to be released

Note: The link of the device will be released only after all previous FUM messages to this device were successfully sent. If no pending device, the link will be released immediately.

4.1.16 Keep alive

Message sends from CMBS to application in order to check connection.

Syntax:

KEEP_ALIVE

Parameters:

None.

4.1.17 Keep alive response

Syntax:

KEEP_ALIVE_RES

Parameters:

None.

4.1.18 Get Target State Request

General: This message can be used in order to verify target is up. Target state is checked periodically and saved. When this request is received, the last saved state is sent to the client.

Syntax:

GET_TARGET_STATE

Parameters:

None.

4.1.19 Target State Change Indication/ Get Target State Response

Message sent from CMBS to application in order notify about a change in target status or as a response for getting target state message which was requested by the client.

Syntax:

TARGET_STATE

STATE: UP / DOWN

Parameters:

UP or DOWN.

4.1.20 Error indication

Message sends from CMBS to application in order to notify about errors.

Example of errors:

- CMBS supports a defined number of clients (MAX_MULTIPLE_CLIENTS), if new application tries to connect CMBS after reached this number, an error indication sends.
- CMBS supports a defined number of messages per client (MAX_CLIENT_REQUEST_PER_MESSAGE), if application tries to send messages after reached this number, and error indication sends.

Syntax:

ERROR

TOO_MANY_CLIENTS / TOO_MANY_MESSAGES

Parameters:

TOO_MANY_CLIENTS or TOO_MANY_MESSAGES.

4.1.21 Get Device Table Phase 2 Request

General: Read HAN device table by chunks. DEV INDEX is the index in the table (not the DEV DECT ID).

Application can read the table from DEV_INDEX 0 by chunks of 5 devices each time, till the response is less than 5 devices. Then application knows that the table is fully read.

Syntax:

GET_DEV_TABLE

DEV_INDEX: X

HOW_MANY: Y

Parameters:

DEV_INDEX – An index (of HAN table) of the first required device (start with 0)

HOW_MANY – How many devices to return

4.1.22 Device Table Phase 2 Response

The response sends number of devices in this response. Each device has its DECT ID, IPUI, and number of UNITS. For each UNIT, it has its type and number of INTERFACEs. For each INTERFACE, it has a TYPE (server or client) and an ID.

Note: Mandatory UNITSs (as UNIT 0) and mandatory INTERFACEs (for each UNIT type) are not listed.

Note: The indentation below is for illustration needs only. All lines should start with a single blank only.

Syntax:

```
DEV_TABLE
DEV_INDEX:  0
NO_OF_DEVICES: 2
DEV_ID:    2
        DEV_IPUI:  0 254 85 8 0
        DEV_EMC:  250 16
        ULE_CAPABILITIES: 5
        ULE_PROTOCOL_ID: 1
        ULE_PROTOCOL_VERSION: 2
        NO_UNITS:         1
        UNIT_ID:  3
        UNIT_TYPE: 7
                NO_OF_INTRF:  1
                INTRF_TYPE:     0 / 1 (server / client)
                INTRF_ID:         256
DEV_ID:5
        DEV_IPUI:  0 253 84 8 1
        DEV_EMC:  250 16
        ULE_CAPABILITIES: 5
        ULE_PROTOCOL_ID: 1
        ULE_PROTOCOL_VERSION: 2
        NO_UNITS:         2
        UNIT_ID:    3
        UNIT_TYPE: X
                NO_OF_INTRF:  3
                INTRF_TYPE:     0 / 1 (server / client)
                INTRF_ID:         256
               INTRF_TYPE:     0 / 1 (server / client)
               INTRF_ID:         257
               INTRF_TYPE:     0 / 1 (server / client)
               INTRF_ID:         258
        UNIT_ID:    4
        UNIT_TYPE: Y
               NO_OF_INTRF:  2
               INTRF_TYPE:     0 / 1 (server / client)
               INTRF_ID:         256
               INTRF_TYPE:     0 / 1 (server / client)
                INTRF_ID:         257
```

Etc.

Parameters:

DEV_INDEX – index (in HAN table) of first device of this message (start with 0).

NO_OF_DEVICES – Number of Devices in this message (currently not including handsets)

Now comes the list of the devices.

For each device:

DEV_ID – unique DECT device ID

DEV_IPUI – IPUI (DECT mac) of the device (5 bytes)

DEV_EMC – EMC (DECT manufactory code) of the device (2 bytes)

ULE_CAPABILITIES – capabilities supported by the device (1 byte):

- 0 – ULE capabilities not supported
- 1 – ULE phase 1 capabilities
- 3 – ULE phase 1.2 capabilities
- 5 – ULE phase 2 capabilities
- 7 – ULE phase 3 capabilities

ULE_PROTOCOL_ID – protocol id implemented by the device (1 byte):

- 0 – Protocol id undefined
- 1 – ULE protocol 1
- 2 – ULE protocol 2

ULE_PROTOCOL_VERSION - protocol version implemented by the device (1 byte):

- 0 - Protocol version undefined
- 1 – protocol version 1.0
- 2 – protocol version 1.4

NO_UNITS – how many non-mandatory units the device supports (generally manager unit + 1).

UNIT_ID – the ID of this UNIT in that device

UNIT_TYPE – unit type as defined in FUN standard.

NO_OF_INTRF – Number of non mandatory INTERFACES in this UNIT.

INTRF_TYPE -- If server (0) or Client (1)

INTRF_ID – the proprietary INTERFACE ID.

<Possible Additional interfaces for this unit>

<Possible Additional units for this device>

4.1.23 Get Black List Device Table Request

General: Read HAN black list device table by chunks. DEV INDEX is the index in the table (not the DEV DECT ID).

Application can read the table from DEV_INDEX 0 by chunks of 5 devices each time, till the response is less than 5 devices. Then application knows that the table is fully read.

Syntax:

GET_BLACK_LIST_DEV_TABLE

DEV_INDEX: X

HOW_MANY: Y

Parameters:

DEV_INDEX – An index (of HAN table) of the first required device (start with 0)

HOW_MANY – How many devices to return

4.1.24 Device Black List Table Response

The response sends number of devices in this response. Each device has its DECT ID, IPUI, and number of UNITS. For each UNIT, it has its type and number of INTERFACEs. For each INTERFACE, it has a TYPE (server or client) and an ID.

Note: Mandatory UNITSs (as UNIT 0) and mandatory INTERFACEs (for each UNIT type) are not listed.

Note: The indentation below is for illustration needs only. All lines should start with a single blank only.

Syntax:

```
BLACK_LIST_DEV_TABLE
 DEV_INDEX: 0
 NO_OF_DEVICES: 2
 DEV_ID:  2
        DEV_IPUI: 0 254 85 8 0
        DEV_EMC: 250 16
        ULE_CAPABILITIES: 5
        ULE_PROTOCOL_ID: 1
        ULE_PROTOCOL_VERSION: 2
        NO_UNITS:       1
        UNIT_ID: 3
        UNIT_TYPE: 7
                NO_OF_INTRF: 1
                INTRF_TYPE:   0 / 1 (server / client)
                INTRF_ID:     256
 DEV_ID:        5
        DEV_IPUI: 0 253 84 8 1
        DEV_EMC: 250 16
        ULE_CAPABILITIES: 5
        ULE_PROTOCOL_ID: 1
        ULE_PROTOCOL_VERSION: 2
        NO_UNITS:       2
        UNIT_ID:  3
        UNIT_TYPE: X
                NO_OF_INTRF: 3
                INTRF_TYPE:   0 / 1 (server / client)
                INTRF_ID:     256
               INTRF_TYPE:   0 / 1 (server / client)
```

INTRF_ID: 257
INTRF_TYPE: 0 / 1 (server / client)
INTRF_ID: 258
UNIT_ID: 4
UNIT_TYPE: Y
NO_OF_INTRF: 2
INTRF_TYPE: 0 / 1 (server / client)
INTRF_ID: 256
INTRF_TYPE: 0 / 1 (server / client)
INTRF_ID: 257

Etc.

Parameters:

DEV_INDEX – index (in HAN table) of first device of this message (start with 0).

NO_OF_DEVICES – Number of Devices in this message (currently not including handsets)

Now comes the list of the devices.

For each device:

DEV_ID – unique DECT device ID

DEV_IPUI – IPUI (DECT mac) of the device (5 bytes)

DEV_EMC – EMC (DECT manufactory code) of the device (2 bytes)

ULE_CAPABILITIES – capabilities supported by the device (1 byte):

- 0 – ULE capabilities not supported
- 1 – ULE phase 1 capabilities
- 3 – ULE phase 1.2 capabilities
- 5 – ULE phase 2 capabilities
- 7 – ULE phase 3 capabilities

ULE_PROTOCOL_ID – protocol id implemented by the device (1 byte):

- 0 – Protocol id undefined
- 1 – ULE protocol 1
- 2 – ULE protocol 2

ULE_PROTOCOL_VERSION - protocol version implemented by the device (1 byte):

- 0 - Protocol version undefined
- 1 – protocol version 1.0
- 2 – protocol version 1.4

NO_UNITS – how many non-mandatory units the device supports (generally manager unit + 1).

UNIT_ID – the ID of this UNIT in that device

UNIT_TYPE – unit type as defined in FUN standard.

NO_OF_INTRF – Number of non mandatory INTERFACES in this UNIT.

INTRF_TYPE -- If server (0) or Client (1)

INTRF_ID – the proprietary INTERFACE ID.

<Possible Additional interfaces for this unit>

<Possible Additional units for this device>

4.2 FUN messages

4.2.1 Send a FUN message to / from a HAN device

Syntax:

FUN_MSG

SRC_DEV_ID: 1

SRC_UNIT_ID: 1

DST_DEV_ID: 0

DST_UNIT_ID: 2

DEST_ADDRESS_TYPE: 0 only for now

MSG_TRANSPORT: 0 only

MGS_SEQ: (cookie) if 0 -- no cookie

MSGTYPE: 1

INTRF_TYPE: 0 / 1 (server / client)

INTRF_ID: 256

INTRF_MEMBER: (old name: CMD_ATTRID)

DATALEN: 6

DATA: 1 F 13 AB 5 6 (hex bytes, see below)

Parameters:

SRC_DEV_ID: 1	
SRC_UNIT_ID: 1	
DST_DEV_ID: 0	
DST_UNIT_ID: 2	
DEST_ADDRESS_TYPE:	For now: 0 only
MSG_TRANSPORT:	For now: 0 only
MGS_SEQ:	Cookie, if needed. Otherwise: 0 (=No cookie)
MSGTYPE: 1	
INTRF_TYPE: 0 / 1 (server / client)	
INTRF_ID: 256	
INTRF_MEMBER:	(old name: CMD_ATTRID), Integer
DATALEN: 6	Number of Additional bytes or 0
DATA: 1 F 13 AB 5 6	in case DATALEN > 0:
	Additional hex bytes

4.2.2 FUN Message Response

Syntax:

FUN_MSG_RES

STATUS: SUCCEED\ FAIL

DEV_ID: x

Parameters:

STATUS: response status - SUCCEED / FAIL

DEV_ID – the device DECT ID to be investigated

4.3 Service messages

4.3.1 Get EEPROM Bytes

Syntax:

[SRV]

GET_EEPROM_BYTES

OFFSET: X (decimal)

SIZE: Y (number of bytes)

Parameters:

OFFSET:EEPROM location to start reading from

SIZE: number of bytes to read

4.3.2 Get EEPROM Bytes Response

Syntax:

[SRV]

GET_EEPROM_BYTES_RES

STATUS: SUCCEED/ FAIL

OFFSET: X (decimal)

SIZE: Y (number of bytes)

DATA: Z (bytes read)

Parameters:

STATUS: response status - SUCCEED / FAIL

OFFSET:EEPROM location to start reading from

SIZE: number of bytes to read

DATA: EEPROM bytes read in case of success, received as list of bytes without delimiter.

4.3.3 Set EEPROM Bytes

Syntax:

[SRV]

SET_EEPROM_BYTES

OFFSET: X (decimal)

DATA: Y (bytes to write)

Parameters:

OFFSET:EEPROM location to start writing to

DATA: EEPROM bytes to write, sent as list of bytes without delimiter.

4.3.4 Set EEPROM Bytes Response

Syntax:

[SRV]

SET_EEPROM_BYTES_RES

STATUS: SUCCEED/ FAIL

Parameters:

STATUS: response status - SUCCEED / FAIL

4.3.5 Get EEPROM Parameter

Syntax:

[SRV]

GET_EEPROM_PARAM

NAME: RFPI

Parameters:

NAME: EEPROM parameter name from list g_EepromParamName described in appendix A.

4.3.6 Get EEPROM Parameter Response

Syntax:

[SRV]

GET_EEPROM_PARAM_RES

STATUS: SUCCEED / FAIL

NAME: RFPI

DATA: 00feb06920

Parameters:

STATUS: response status - SUCCEED / FAIL

NAME: EEPROM parameter name from list g_EepromParamName described in appendix A.

DATA: EEPROM parameter value read in case of success, received as list of bytes without delimiter.

4.3.7 Set EEPROM Parameter

Syntax:

[SRV]

SET_EEPROM_PARAM

NAME: RFPI

DATA: 00feb06920

Parameters:

NAME: EEPROM parameter name from list g_EepromParamName described in appendix A.

DATA: EEPROM parameter value to write, sent as list of bytes without delimiter.

4.3.8 Set EEPROM Parameter Response

Syntax:

[SRV]

SET_EEPROM_PARAM_RES

STATUS: SUCCEED / FAIL

NAME: RFPI

Parameters:

STATUS: response status - SUCCEED / FAIL

NAME: EEPROM parameter name from list g_EepromParamName described in appendix A.

4.3.9 Get Production Parameter

Syntax:

[SRV]

GET_PRODUCTION_PARAM

NAME: DECT_TYPE

Parameters:

NAME: Production parameter name from list g_ProdParamName described in appendix A.

4.3.10 Get Production Parameter Response

Syntax:

[SRV]

GET_ PRODUCTION _PARAM_RES

STATUS: SUCCEED / FAIL

NAME: DECT_TYPE

DATA: 00

Parameters:

STATUS: response status - SUCCEED / FAIL

NAME: Production parameter name from list g_ProdParamName described in appendix A.

DATA: Production parameter value read in case of success, received as list of bytes without delimiter.

4.3.11 Set Production Parameter

Syntax:

[SRV]

SET_ PRODUCTION _PARAM

NAME: DECT_TYPE

DATA: 00

Parameters:

NAME: Production parameter name from list g_ProdParamName described in appendix A.

DATA: Production parameter value to write, sent as list of bytes without delimiter.

4.3.12 Set Production Parameter Response

Syntax:

[SRV]

SET_ PRODUCTION _PARAM_RES

STATUS: SUCCEED / FAIL

NAME: DECT_TYPE

Parameters:

STATUS: response status - SUCCEED / FAIL

NAME: Production parameter name from list g_ProdParamName described in appendix A.

4.3.13 Get EEPROM Size

Syntax:

[SRV]

GET_EEPROM_SIZE

Parameters:

None.

4.3.14 Get EEPROM Size Response

Syntax:

[SRV]

GET_EEPROM_SIZE_RES

SIZE: 9208

Parameters:

SIZE: EEPROM size in bytes

4.3.15 Set RF State

Syntax:

[SRV]

SET_RF_STATE

STATE: RESUMED/ SUSPENDED

Parameters:

STATE: required state - RESUMED/ SUSPENDED

4.3.16 Get Target Hardware Version

Syntax:

[SRV]

GET_TARGET_HW_VERSION

Parameters:

None.

4.3.17 Get Target Hardware Version Response

Syntax:

[SRV]

GET_TARGET_HW_VERSION_RES

STATUS: SUCCEED

HW_CHIP: HW_CHIP_DCX81

HW_CHIP_VERSION: HW_CHIP_VERSION_C

HW_BOARD: HW_BOARD_MOD

HW_COM_TYPE: HW_COM_TYPE_UART

Parameters:

STATUS: response status - SUCCEED / FAIL

HW_CHIP: one of the following values: HW_CHIP_VEGAONE, HW_CHIP_DCX78, HW_CHIP_DCX79, HW_CHIP_DCX81, HW_CHIP_DVF99, HW_CHIP_DHX91

HW_CHIP_VERSION: one of the following values: HW_CHIP_VERSION_B, HW_CHIP_VERSION_C, HW_CHIP_VERSION_D

HW_BOARD: one of the following values: HW_BOARD_MOD, HW_BOARD_DEV

HW_COM_TYPE: one of the following values: HW_COM_TYPE_UART, HW_COM_TYPE_USB, HW_COM_TYPE_SPI0, HW_COM_TYPE_SPI3

4.3.18 Get Software Version

Syntax:

[SRV]

GET_SW_VERSION

Parameters:

None.

4.3.19 Get Software Version Response

Syntax:

[SRV]

GET_SW_VERSION_RES STATUS: SUCCEED

STATUS: SUCCEED

TARGET_VERSION: 406

TARGET_BUILD_NUMBER: 23

HOST_VERSION: 406

HOST_BUILD_NUMBER: 23

Parameters:

STATUS: response status - SUCCEED / FAIL

TARGET_VERSION: target version value in hex

TARGET_BUILD_NUMBER: target build number value in hex

HOST_VERSION: host version value in hex

HOST_BUILD_NUMBER: host build number value in hex

4.3.20 Reset Target

Syntax:

[SRV]

RESET_TARGET

Parameters:

None.

4.3.21 Get Number of Registered Devices

Syntax:

[SRV]

GET_NUM_OF_REG_DEVICES

Parameters:

None

4.3.22 Get Number of Registered Devices Response

Syntax:

[SRV]

GET_NUM_OF_REG_DEVICES_RES

NUM_REG_DEV: 1

Parameters:

NUM_REG_DEV:	Number of devices registered to the base.

4.3.23 Get Maximum Number of Devices

Syntax:

[DBG]

GET_MAX_NUM_OF_DEVICES

Parameters:

None

4.3.24 Get Maximum Number of Devices Response

Syntax:

[DBG]

GET_MAX_NUM_OF_DEVICES_RES

MAX_NUM_DEV: 30

Parameters:

MAX_NUM_DEV	Maximum number of devices supported

4.3.25 Set NEMo

Syntax:

[SRV]

SET_NEMO_MODE

MODE: x

Parameters:

MODE: NEMO mode parameter from list g_SetNEMoModeList described in appendix A.

4.3.26 Set NEMo Response

Syntax:

[SRV]

SET_NEMO_MODE_RES

STATUS: SUCCESS/ FAIL

Parameters:

STATUS: response status - SUCCEED / FAIL

4.3.27 Initiate Firmware Update

Syntax:

[SRV]

FIRMWARE_UPDATE_START

FILE_NAME: filename

PACKET_SIZE: X(number of bytes)

RESTORE_PARAM: 0/1

Parameters:

FILE_NAME: Firmware binary file name for update.

PACKET_SIZE: Packet size to update. Possible sizes- 32, 64, 128, 256, 512 bytes

RESTORE_PARAM: Restore EEPROM parameters after firmware upgrade.

0 – No

1 – Yes

4.3.28 Initiate Firmware Update Response

Syntax:

[SRV]

FIRMWARE_UPDATE_START_RES

STATUS: SUCCESS/ FAIL

Parameters:

STATUS: response status - SUCCESS / FAIL

4.3.29 End of Firmware Update Indication

Syntax:

[SRV]

FIRMWARE_UPDATE_END_RES

STATUS: SUCCESS/ FAIL

Parameters:

STATUS: response status - SUCCESS / FAIL

4.4 Debug Messages

4.4.1 Get RAM Bytes

Syntax:

[DBG]

GET_RAM_BYTES

ADDRESS: X (decimal)

SIZE: Y (number of bytes)

Parameters:

ADDRESS: RAM address to start reading from

SIZE: number of bytes to read

4.4.2 Get RAM Bytes Response

Syntax:

[DBG]

GET_RAM_BYTES_RES

STATUS: SUCCEED / FAIL

ADDRESS: X (decimal)

SIZE: Y (number of bytes)

DATA: Z (bytes read)

Parameters:

STATUS: response status - SUCCEED / FAIL

ADDRESS: RAM address to start reading from

SIZE: number of bytes to read

DATA: RAM bytes read in case of success, received as list of bytes without delimiter.

4.4.3 Set RAM Bytes

Syntax:

[DBG]

SET_RAM_BYTES

ADDRESS: X (decimal)

DATA: Y (bytes to write)

Parameters:

ADDRESS: RAM location to start writing to

DATA: RAM bytes to write, sent as list of bytes without delimiter.

4.4.4 Set RAM Bytes Response

Syntax:

[DBG]

SET_RAM_BYTES_RES

STATUS: SUCCEED / FAIL

Parameters:

STATUS: response status - SUCCEED / FAIL

4.4.5 Get Number of Messages in FUN MSG Q

Syntax:

[DBG]

GET_NUM_OF_FUN_MSG_IN_Q

DEV_ID: x

Parameters:

DEV_ID: message queue indicated by DEV_ID

4.4.6 Get Number of Messages in FUN MSG response

Syntax:

[DBG]

GET_NUM_OF_FUN_MSG_IN_Q_RES

STATUS: SUCCEED/ FAIL

DEV_ID: x

NUM_OF_PENDING_MSG: y

MAX_MSG_IN_Q: z

Parameters:

STATUS: response status - SUCCEED / FAIL

DEV_ID: message queue indicated by DEV_ID

NUM_OF_PENDING_MSG: number of pending message in the queue

MAX_MSG_IN_Q: message queue capacity

4.4.7 Clear Messages in FUN MSG Q

Syntax:

[DBG]

CLEAR_FUN_MSG_Q

DEV_ID: x Parameters:

Parameters:

DEV_ID: message queue indicated by DEV_ID

4.5 Call Messages

4.5.1 Call Established Indication

Syntax:

[CALL]

CALL_ESTABLISH_INDICATION

CALL_ID: 0

PARTICIPANT: D0001

Parameters:

CALL_ID: Call id allocated by cmbs_tcx

PARTICIPANT: Device as “DXXXX” where XXXX is the 2 byte device id

HS as HS number, for example: 1

4.5.2 Device Released from Call

Syntax:

[CALL]

DEV_RELEASED_FROM_CALL

CALL_ID: 0

DEV_ID: D0001

Parameters:

CALL_ID: Call id allocated by cmbs_tcx

DEV_ID:Device as “DXXXX” where XXXX is the 2 byte device id

4.5.3 HS Released from Call

Syntax:

[CALL]

HS_RELEASED_FROM_CALL

CALL_ID: 0

HS_NUMBER: 1

Parameters:

CALL_ID: Call id allocated by cmbs_tcx

HS_NUMBER: HS number

4.5.4 Merge Calls Indication

Syntax:

[CALL]

MERGE_CALLS_INDICATION

CALL_ID: 0

PARTICIPANT: D0001

PARTICIPANT: D0002

Parameters:

CALL_ID: Call id of the merged call

PARTICIPANT: Device as “DXXXX” where XXXX is the 2 byte device id

HS as HS number, for example: 1

4.5.5 Call Released Indication

Syntax:

[CALL]

CALL_RELEASE_INDICATION

CALL_ID: 0

Parameters:

CALL_ID: Call id of the released call

4.5.6 Call Release

Syntax:

[CALL]

CALL_RELEASE

CALL_ID: 0

Parameters:

CALL_ID: Call id of the call to be released

4.5.7 Get All Active Calls Information

Syntax:

[CALL]

GET_ACTIVE_CALLS_INFO

Parameters:

None

4.5.8 Get All Active Calls Information Response

Note: The indentation below is for illustration needs only. All lines should start with a single blank only.

Syntax:

[CALL]

GET_ACTIVE_CALLS_INFO_RES

NUM_OF_ACTIVE_CALLS: 2

CALL_ID: 1

LINE_ID: 1

CALL_STATE: CALL_ACTIVE

PARTICIPANT: 1

CALL_ID: 0

LINE_ID: 0

CALL_STATE: CALL_ACTIVE

PARTICIPANT: 2

Parameters:

NUM_OF_ACTIVE_CALLS: Total number of active calls present

CALL_ID: Call ID allocated

LINE_ID: Line ID allocated

PARTICIPANT: Device as “DXXXX” where XXXX is the 2 byte device id

HS as HS number, for example: 1

CALL_STATE: State of the ongoing call. The following are the call states that are supported:

"CALL_CLOSE";

"CALL_INC_PEND";

"CALL_INC_RING";

"CALL_OUT_PEND";

"CALL_OUT_PEND_DIAL";

"CALL_OUT_INBAND";

"CALL_OUT_PROC";

"CALL_OUT_RING";

"CALL_ACTIVE";

"CALL_RELEASE";

"CALL_ON_HOLD";

"CALL_CONFERENCE";

"CALL_SCREENING";

4.6 ULE Data Call Messages

4.6.1 Indication for Data Call Session Created

Data call session created indication from server to client. This shall be send when data call session created as a response to data call setup request by client or when call setup initiated by device.

Syntax:

[DATA_CALL]

SESSION_CREATED_INDICATION

APP_ID: X

SESSION_ID: Y

Parameters:

APP_ID: Data call application ID

SESSION_ID: Data call session ID

4.6.2 Data Call Release Request

Data call release request to cancel data call from client to server.

Syntax:

[DATA_CALL]

RELEASE_REQUEST

SESSION_ID: X

Parameters:

SESSION_ID: Data call session id

4.6.3 Data Call Released Indication

This shall be send from server to client when data call released as a response to data call release request by client or when data call release initiated by device.

Syntax:

[DATA_CALL]

RELEASED_INDICATION

SESSION_ID: X

STATUS: SUCCESS/FAIL

Parameters:

SESSION_ID: Data call session ID

STATUS: Release status – SUCCESS or FAIL

4.6.4 Send Data

Request from client to send data through data call.

Syntax:

[DATA_CALL]

SEND_DATA

SESSION_ID: X

DATALEN: Y

DATA: Z

Parameters:

SESSION_ID: Data call session ID

DATALEN: Length of data to send

DATA: Data to send

4.6.5 Send Data Acknowledgement

Acknowledgement from server for send data request.

Syntax:

[DATA_CALL]

SEND_DATA_ACK

SESSION_ID: X

STATUS: SUCCESS/FAIL

Parameters:

SESSION_ID: Data call session ID

STATUS: Data send status – SUCCESS or FAIL

4.6.6 Receive Data

Data received indication from server to client.

Syntax:

[DATA_CALL]

RECEIVED_DATA

SESSION_ID: X

DATALEN: Y

DATA: Z

Parameters:

SESSION_ID: Data call session ID

DATALEN: Length of received data

DATA: Received data

5. Flows

5.1 Device Registration Flow

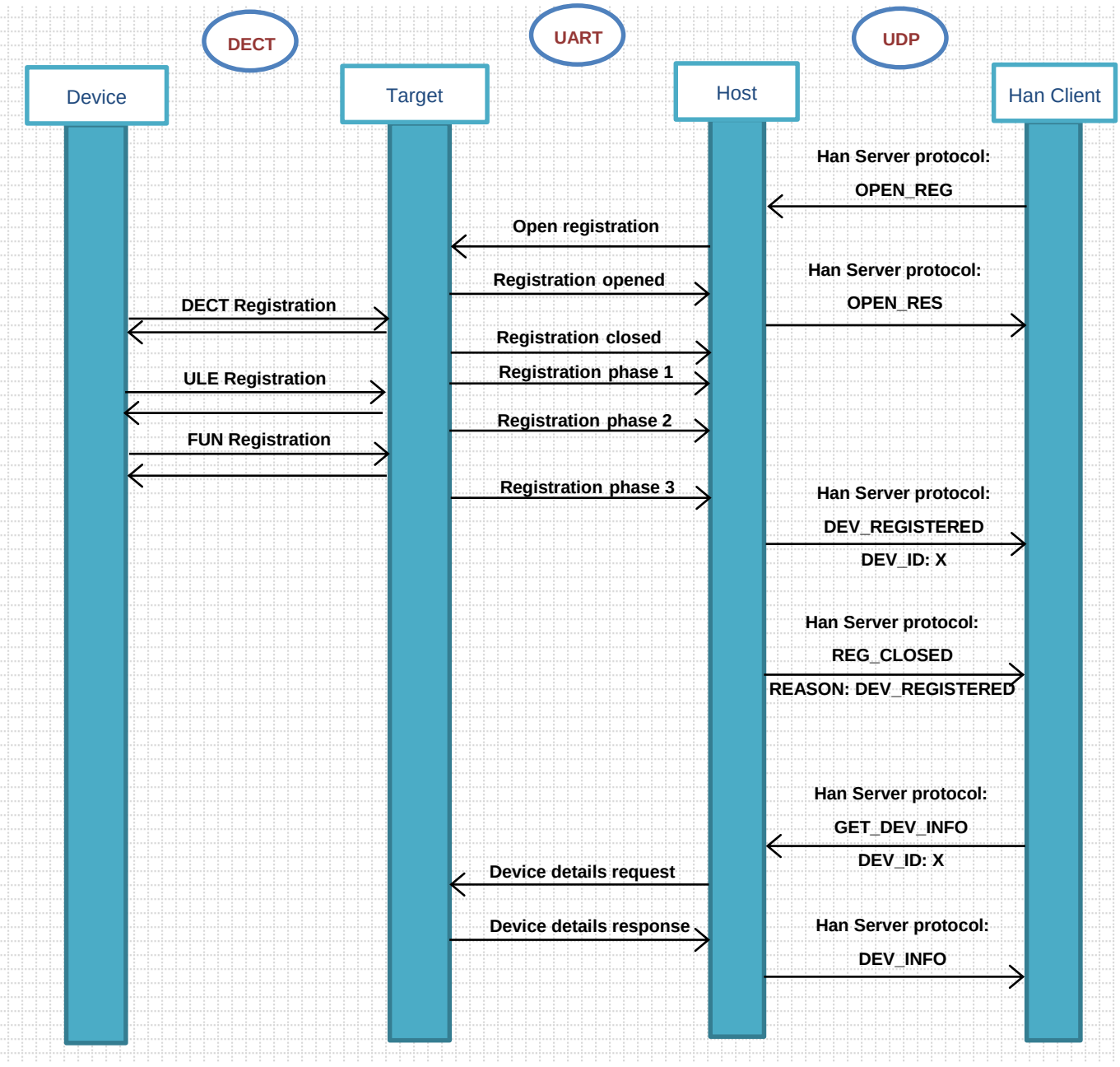


Figure 2- Device Registration Flow

5.2 Single FUN Message Sending Flow

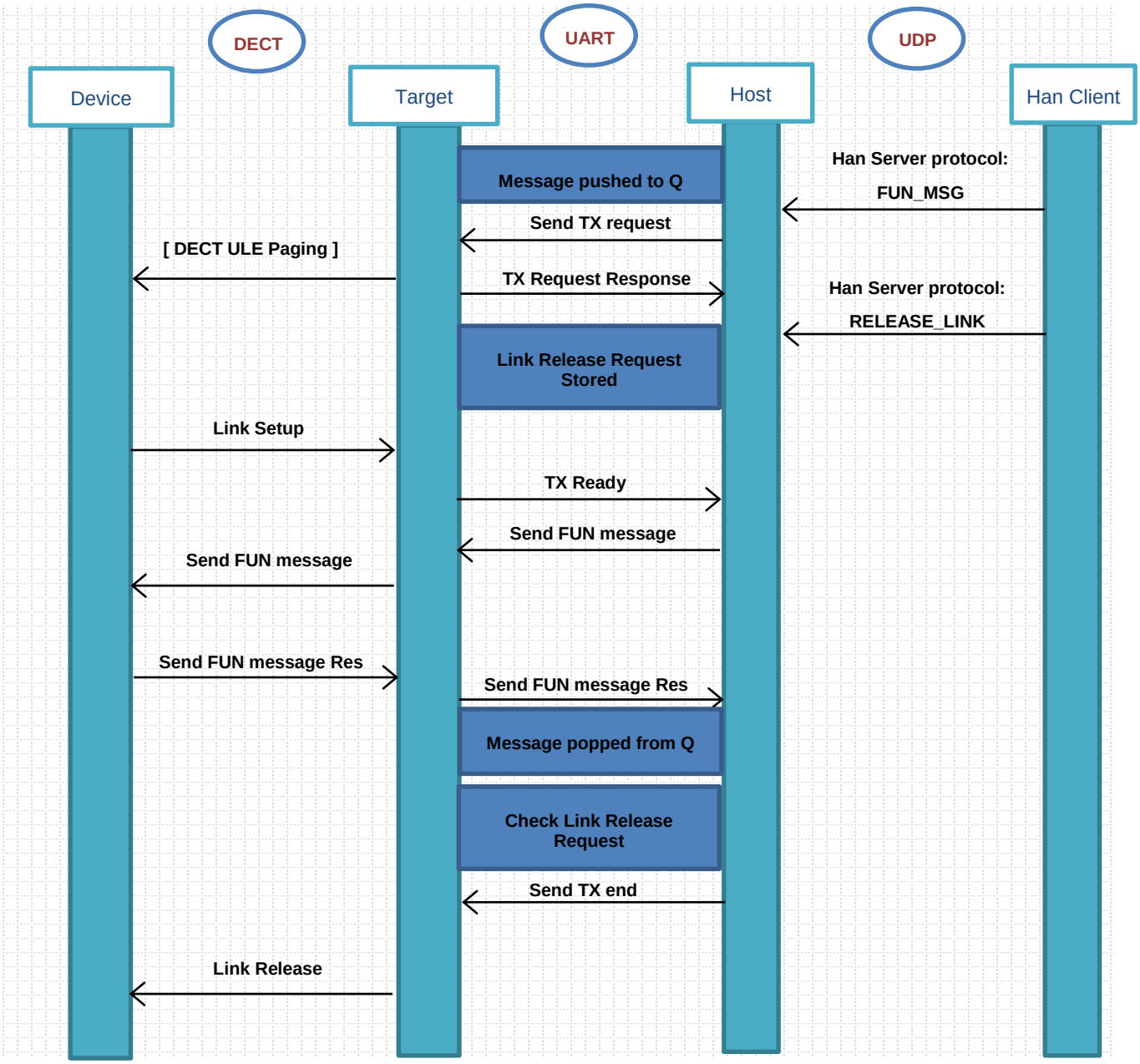


Figure 3 - Single FUN message Sending Flow

5.3 Multiple FUN Message Sending Flow

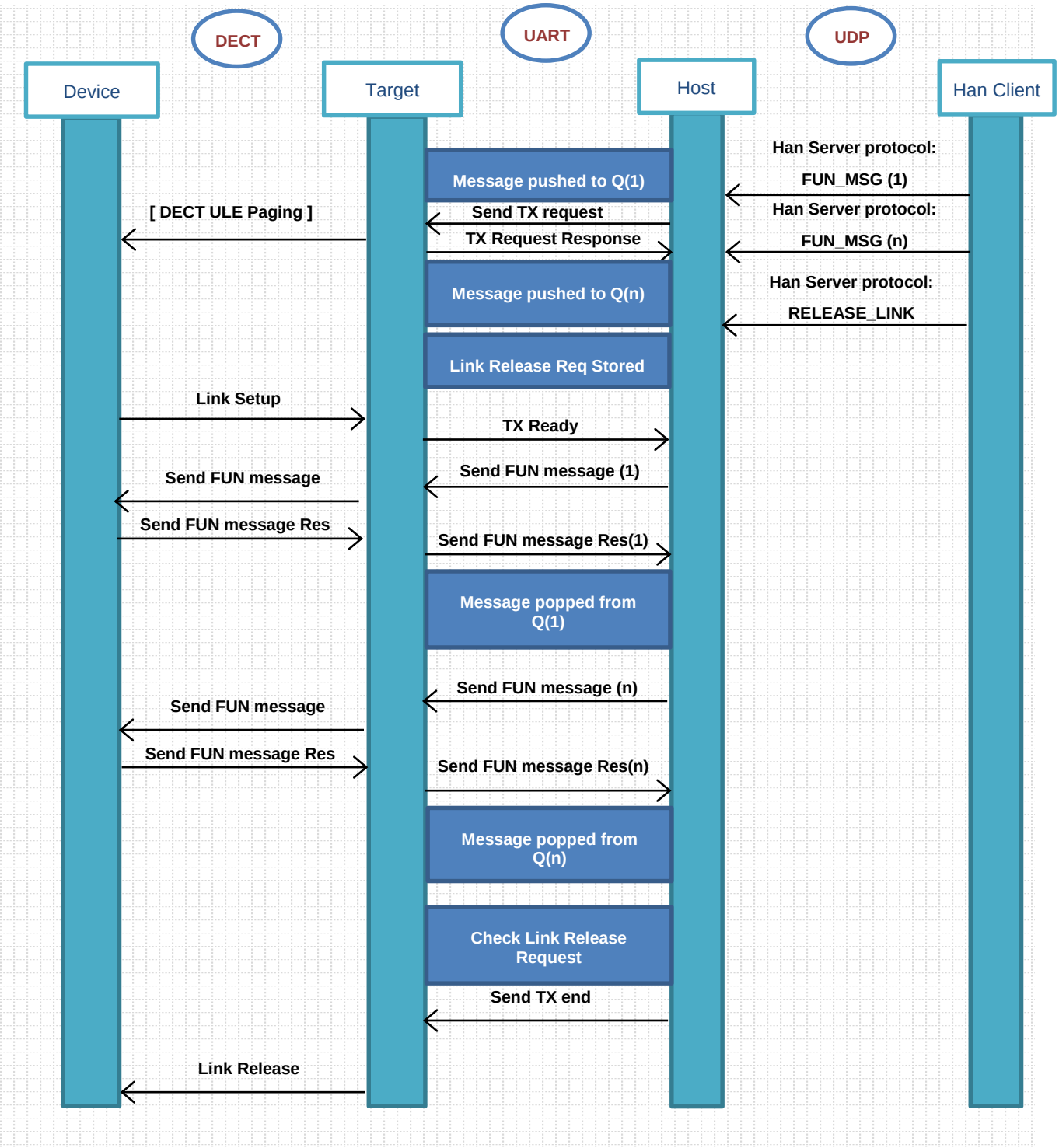


Figure 4 - Multiple FUN Messages Sending

5.4 FUN Message Sending Flow – No Link Release

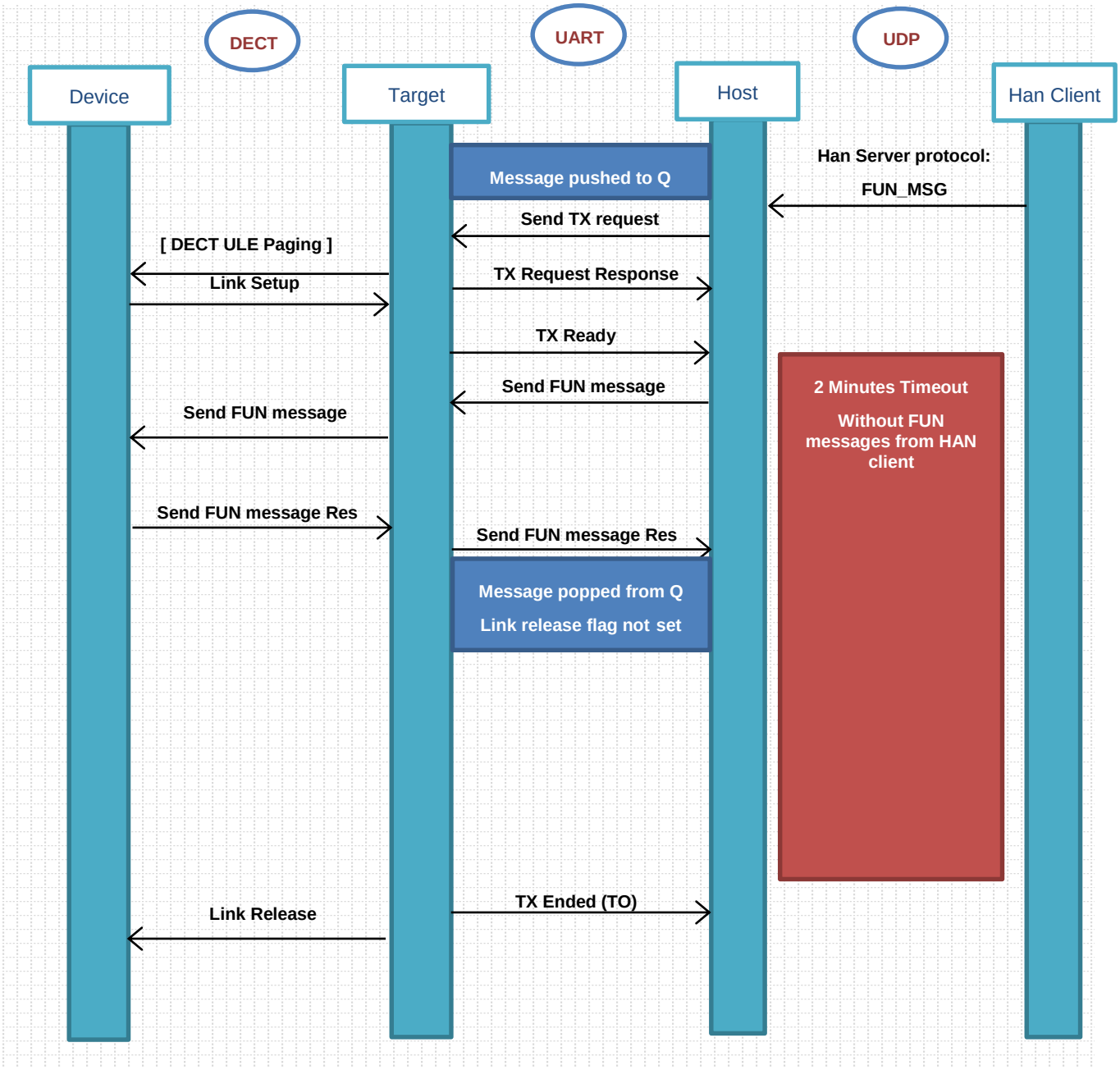


Figure 5 - Fun message sending - no link release

5.5 Read Device Table Flow

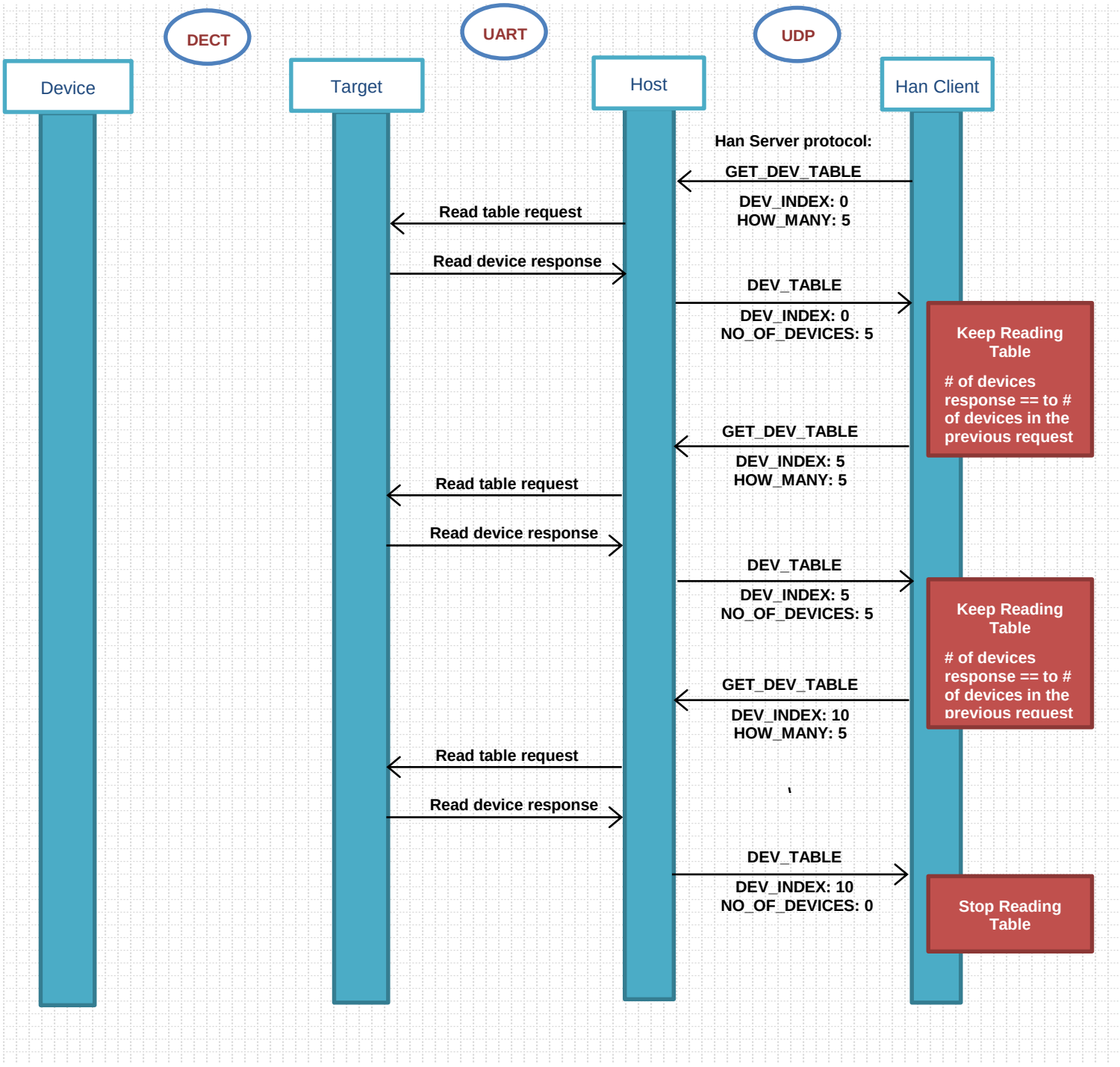


Figure 6 - Read Device Table Flow

5.6 Call from device to device

For this case, let's assume voice call routing table contains one record:
Source: device 1, unit 1

Destination: device 2, unit 1

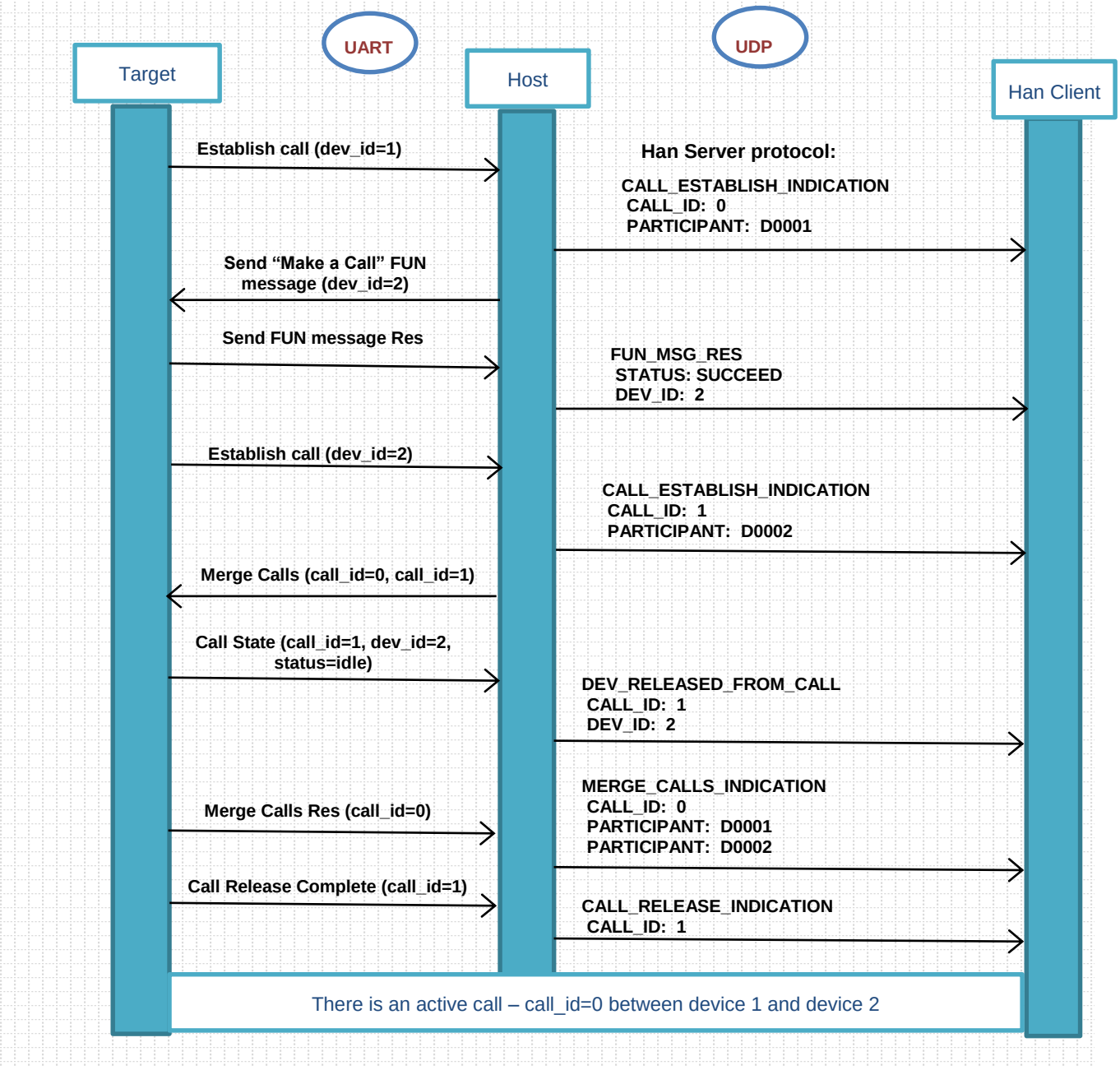


Figure 7 – Call from device to device

Appendix A

```
char *g_EepParamNameList[] =
{
        "RFPI",
        "RXTUN",
        "RF_FULL_POWER",
        "PREAM_NORM",
        "RF19APU_SUPPORT_FCC",
        "RF19APU_DEVIATION",
        "RF19APU_PA2_COMP",
        "MAX_USABLE_RSSI",
        "LOWER_RSSI_LIMIT",
        "PHS_SCAN_PARAM",
        "JDECT_LEVEL1_M82",
        "JDECT_LEVEL2_M62",
        "SUBS_DATA",
        "RVREF",
        "GFSK",
        "HAN_DECT_SUB_DB_START",
        "HAN_DECT_SUB_DB_END",
        "HAN_ULE_SUB_DB_START",
        "HAN_ULE_SUB_DB_END",
        "HAN_FUN_SUB_DB_START",
        "HAN_FUN_SUB_DB_END",
        "HAN_ULE_NEXT_TPUI",
        "MAX_TRANSFER_SIZE",
        "HAN_FUN_GROUP_LIST_START",
        "HAN_FUN_GROUP_LIST_END",
        "HAN_FUN_GROUP_TABLE_START",
        "HAN_FUN_GROUP_TABLE_END",
        "HAN_ULE_BROADCAST_CONVERSION_TABLE_START",
        "HAN_ULE_BROADCAST_CONVERSION_TABLE_END",
        "ULE_MULTICAST_ENC_PARAMS",

   NULL
};

char *g_SetNEMoModeList[] =
{
        "OFF",
        "ON",
    NULL
};
```