MULTIPURPOSE COMMUNOCATION PROTOCOL BETWEEN ERA GLONASS VEHICLE TERMINAL AND OPERATOR FRAMEWORK. PROTOCOL REFERENCE INMPLEMENTATION TESTS – EMERGENCY SERVICES CALL

SCH OKR «ERA GLONASS»
(Developmental work)

Testing Order and Methods

16 Pages

2011

Contents

1. TEST ITEM

1.1 The item under test is a source code of communication protocol between the vehicle system “ERA GLONASS” and the Operator Framework, further referred to as EGTS Protocol.

The source code is used to standardize the process of developing the terminal devices of “ERA GLONASS” system.

1.2. Cipher: СЧ ОКР «ЭРА ГЛОНАСС» (developmental work “Era Glonass”)

2. TESTING PURPOSE

2.1. Checking if the source code of the Transport Level Protocol Reference Implementation corresponds to the “Terminal Era Glonass. Communication Protocol. Transport Level. Version 1.0” specification.

2.2 Checking is the source code of the Service Support Protocol Reference Implementation corresponds to the “Terminal Era Glonass. Communication Protocol. Service Support Protocol. Version 1.0” specification.

2.3 Checking the source code compatibility with validation tool provided by protocol specification developers.

3. PROGRAM REQUIREMENTS

3.1. Functioning of EGTS Protocol should correspond to the multipurpose protocol requirements mentioned in the requirements specification “Development of Multipurpose Communication Protocol between vehicle terminal “Era Glonass” and Operator Framework in the part of development and testing of the reference implementation of the protocol – Emergency Services Call”.

4. REQUIREMENTS TO THE PROGRAM DOCUMENTS

Tests are done on the basis of the following directive documents:

-- “Vehicle System ERA GLONASS. Binding Technical Reqirements” (version 1.12);

– Description of the source code of “Multipurpose Communication Protocol between vehicle terminal “Era Glonass” and Operator Framework in the part of testing of the reference implementation of the protocol – Emergency Services Call”;

- Requirements «Terminal ERA GLONASS. Communication Protocol. Transport Level. Version 1.0”;

- Requirements “Terminal Era Glonass. Communication Protocol. Service Support Protocol. Version 2.0” ;

– Program and testing methods of “Multipurpose Communication Protocol between vehicle terminal “Era Glonass” and Operator Framework in the part of testing of the reference implementation of the protocol – Emergency Services Call”;

- Technical Requirements “Development of Multipurpose Communication Protocol between vehicle terminal “Era Glonass” and Operator Framework in the part of development and testing of the reference implementation of the protocol – Emergency Services Call”;

- Regulations on the program code preparation.

5. MEANS AND ORDER OF TESTS

5.1. The tests are done by a board where the experts of CJSC “Santel-Navigatsiya” are included. The board has the right to precise the amount, order and methods of testing.

5.2. Valuating the results of tests, checks and measurements is done through comparing them with the articles of Technical Requirements for each kind of tests. Multipurpose Communication Protocol between vehicle terminal “Era Glonass” and Operator Framework in the part of testing of the reference implementation of the protocol – Emergency Services Call is said to bare the tests (checks) if it corresponds to the Technical Requirements.

5.3. The tests are done using a personal computer (PC).

PC should correspond to the following hardware and software requirements:

- Chipset o the Pentium IV class;
- Chipset frequency not lower than 1,5 GHz;
- RAM no less than 256 Mb;
- Operating System Windows XP SP2 (SP3);

5.4. Order and sequence of tests is shown in Table 1.

Table 1

№	Tests and Checks	Number of paragraphs of Technical Requirements	Reference in this document
1	Testing the source code of the Transport level	4.I	6.1
2	Testing of the source code of the service “Configuration of equipment and Service control”	4.II.1	6.3
3	Testing of the source code of the service “Emergency Accident Response” in the part of packaged data exchange	4.II.2	6.4
4	Testing of the source code of the service “Emergency Accident Response” in the part of an emergency SMS channel	4.II.2	6.2, 6.5
5	Testing of the source code of the service “Emergency Accident Response” in the part of transmitting the acceleration	4.II.2	6.4

	pattern in case of a road accident		
6	Testing the source code in the part of control and data exchange via SMS	4.II.3	6.2, 6.6
7	Testing the source code in the part of testing and diagnostics of Vehicle Terminal	4.II.4	6.7
8	Testing the source code in the part of downloading the software	4.II.5	6.8
9	Requirements to the reliability of technical means and software	3.1.4.3, 3.1.10.1.2, 3.1.10.1.3	6.9
10	Requirements for the source code to corresponding the standard ISO/IEC 9899:1999.	3.1.10.17	6.10
11	Testing of the source code of the service “Configuration of equipment and Service control” using test vectors and validation tools.	4.II.1	6.11
12	Testing of the source code of the service “Emergency Accident Response” in the part of transmitting the acceleration pattern in case of a road accident, using test vectors and validation tools.	4.II.2	6.12
13	Testing the source code in the part of testing and diagnostics of Vehicle Terminal, using test vectors and validation tools.	4.II.4	6.13
14	Testing the source code in the part of downloading the software using test vectors and validation tools.	4.II.5	6.14

5.5. After the tests, a test protocol should be worked out on every point of methods. It is possible to mention the results of several kinds of tests in one test report.

5.6. Test results are issued in the form of a test certificate.

6. TESTING METHODS

6.1. Source code of Transport Level Tests.

Tests are performed automatically with one launch of egts_transport_probe1.exe test program without the parameters in the Windows command shell. The program performs a few series of sending-receiving different EGST packages. The series are performed under different conditions of package transmitting. The conditions of transmitting vary:

- Header encoding, header field HE;

- Data encoding, for header ENA and SKID. If the specifications for the data encoding algorithm are absent, a test encoding algorithm (gamma xoring + extra bites adding) is used;

- Data compression (Compression), header field CMP. If the specifications for the compression algorithm are absent, a test encoding algorithm (gamma xoring) is used;

-Digital sign-up (Sign-up), packages of the type EGTS_PT_SIGNED_APPDATA. If the specifications for the digital sign-up are absent, a test encoding algorithm (a predefined line as a sign-up) is used;

- packet of the type EGTS_PT_RESPONSE;

Below is given a fragment of the successful test results of one of testing sessions:

```
Header encoding - DBG
Data encoding - NONE
Compression - NONE
Sign-up - DBG
Responce - NONE

AUTH.EGTS_SR_TERM_IDENTITY (simple) ... PASSED
AUTH.EGTS_SR_TERM_IDENTITY (no optional fields) ... PASSED
AUTH.EGTS_SR_TERM_IDENTITY (all optional fields) ... PASSED
AUTH.EGTS_SR_TERM_IDENTITY (max size) ... PASSED
AUTH.EGTS_SR_VEHICLE_DATA (simple) ... PASSED
AUTH.EGTS_SR_AUTH_PARAMS (simple) ... PASSED
AUTH.EGTS_SR_AUTH_PARAMS (no optional fileds) ... PASSED
AUTH.EGTS_SR_AUTH_PARAMS (max size) ... PASSED
AUTH.EGTS_SR_MODULE_DATA (simple) ... PASSED
AUTH.EGTS_SR_AUTH_INFO (simple) ... PASSED
AUTH.EGTS_SR_SERVICE_INFO (simple) ... PASSED
AUTH.EGTS_SR_RESULT_CODE (simple) ... PASSED
COMMANDS.EGTS_SR_COMMAND_DATA (simple COMCONF) ... PASSED
COMMANDS.EGTS_SR_COMMAND_DATA (COMCONF, max length) ... PASSED
COMMANDS.EGTS_SR_COMMAND_DATA (simple COM) ... PASSED
COMMANDS.EGTS_SR_COMMAND_DATA (COM, max length) ... PASSED
FIRMWARE.EGTS_SR_SERVICE_PART_DATA (simple, 64b, first) ... PASSED
FIRMWARE.EGTS_SR_SERVICE_PART_DATA (simple, 64b, second) ... PASSED
```

```
FIRMWARE.EGTS_SR_SERVICE_PART_DATA (simple, max length, first) ...
PASSED
FIRMWARE.EGTS_SR_SERVICE_PART_DATA (simple, max length, second) ...
PASSED
FIRMWARE.EGTS_SR_SERVICE_FULL_DATA (simple, 64b) ... PASSED
FIRMWARE.EGTS_SR_SERVICE_FULL_DATA (simple, max length) ... PASSED
ECALL.EGTS_SR_ACCEL_DATA (simple) ... PASSED
ECALL.EGTS_SR_ACCEL_DATA (max length) ... PASSED
ECALL.EGTS_SR_MSD_DATA (simple) ... PASSED
ECALL.EGTS_SR_MSD_DATA (min length) ... PASSED
ECALL.EGTS_SR_MSD_DATA (max length) ... PASSED
ECALL.EGTS_SR_RAW_MSD_DATA (simple) ... PASSED
ECALL.EGTS_SR_RAW_MSD_DATA (max length) ... PASSED
ECALL.EGTS_SR_TRACK_DATA (min length) ... PASSED
ECALL.EGTS_SR_TRACK_DATA (10 items) ... PASSED
ECALL.EGTS_SR_TRACK_DATA (10 items, gap 5 items , 10 items) ...
PASSED
ECALL.EGTS_SR_TRACK_DATA (all items) ... PASSED
AUTH.EGTS_SR_TERM_IDENTITY+VEHICLE_DATA(simple) ... PASSED
```

Tests are passed successfully if there are no error reports in the program output, for instance:

```
ECALL.EGTS_SR_MSD_DATA (max length) ... frame data too long
FAILED
```

And the output ends with the message

```
erorr count: 0
autotests PASSED
```

6.2 Transport Level of Data transmission via SMS testing

Tests are performed with the help of test program $(EGTS)\probes\sms\get\pdu_get.exe and $(EGTS)\probes\sms\set\pdu_set.exe. For the set of executable files see “Source Code Specification”.

6.2.1 Testing the set of PDU messages for sending

Tests are performed with the help of test program $(EGTS)\probes\sms\set\pdu_set.exe and a set of test files $(EGTS)\probes\sms\set\tests*.test. Test file formats and the description of the testing procedure are given in “Description of the source code”. To perform the test it is necessary to exert pdu_set.exe being under the Windows command shell and in the current directory $(ROOT)\probes\sms\set:

```
pdu_set.exe  <title of the file with the text>
```

Example: ‘`pdu_set.exe .\tests\code100.test`’.

The results of the test performance are kept in the log $(ROOT)\probes\sms\set\set_tests.log. Errors of the test performance are kept in the file $(ROOT)\probes\sms\set\errors.log.

For convenience, the file $(EGTS)\probes\sms\set\run_test.bat is given, it launches the sequence of tests from the list $(EGTS)\probes\sms\set\tests.list.

Tests (a test) are passed successfully if there are no error reports in the file $(ROOT)\probes\sms\set\errors.log.

6.2.2 Testing the parse of PDU messages as received

Tests are performed with the help of test program $(EGTS)\probes\sms\get\pdu_get.exe and a set of test files $(EGTS)\probes\sms\get\tests*.test. Test file formats and the description of the testing procedure are given in “Description of the source code”. To perform the test it is necessary to exert pdu_get.exe being under the Windows command shell and in the current directory $(ROOT)\probes\sms\get:

```
pdu_get.exe  < title of the file with the text >
```

Example: ‘`pdu_get.exe .\tests\code100.test`’.

The results of the test performance are kept in the log $(ROOT)\probes\sms\get\get_tests.log. Errors of the test performance are kept in the file $(ROOT)\probes\sms\get\errors.log.

For convenience, the file $(EGTS)\probes\sms\get\run_test.bat, it launches the sequence of tests from the list $(EGTS)\probes\sms\get\tests.list.

Tests (a test) are passed successfully if there are no error reports in the file $(ROOT)\probes\sms\get\errors.log.

6.3. Testing of the source code of the service “Configuration of equipment and service control”

Testing is performed the same way as in paragraph 6.1. In every test series the packages of the Configuration and Service Control level (service AUTH) are transmitted.

```
AUTH.EGTS_SR_TERM_IDENTITY (simple) ... PASSED
AUTH.EGTS_SR_TERM_IDENTITY (no optional fields) ... PASSED
AUTH.EGTS_SR_TERM_IDENTITY (all optional fields) ... PASSED
AUTH.EGTS_SR_TERM_IDENTITY (max size) ... PASSED
AUTH.EGTS_SR_VEHICLE_DATA (simple) ... PASSED
AUTH.EGTS_SR_AUTH_PARAMS (simple) ... PASSED
AUTH.EGTS_SR_AUTH_PARAMS (no optional fileds) ... PASSED
AUTH.EGTS_SR_AUTH_PARAMS (max size) ... PASSED
AUTH.EGTS_SR_MODULE_DATA (simple) ... PASSED
AUTH.EGTS_SR_AUTH_INFO (simple) ... PASSED
AUTH.EGTS_SR_SERVICE_INFO (simple) ... PASSED
AUTH.EGTS_SR_RESULT_CODE (simple) ... PASSED
```

Tests are passed successfully if there are no error messages in the program output and the output ends with the message

```
erorr count: 0
autotests PASSED
```

.

6.4. Testing of the source code of the service “Emergency Accident Response” in the part of packaged data exchange, including a

transmission of the acceleration pattern in case of a road accident.

Testing is performed the same way as in paragraph 6.1. In every test series the packages of the Emergency Accident Response Level (service ECALL) are transmitted.

```
ECALL.EGTS_SR_ACCEL_DATA (simple) ... PASSED
ECALL.EGTS_SR_ACCEL_DATA (max length) ... PASSED
ECALL.EGTS_SR_MSD_DATA (simple) ... PASSED
ECALL.EGTS_SR_MSD_DATA (min length) ... PASSED
ECALL.EGTS_SR_MSD_DATA (max length) ... PASSED
ECALL.EGTS_SR_RAW_MSD_DATA (simple) ... PASSED
ECALL.EGTS_SR_RAW_MSD_DATA (max length) ... PASSED
ECALL.EGTS_SR_TRACK_DATA (min length) ... PASSED
ECALL.EGTS_SR_TRACK_DATA (10 items) ... PASSED
ECALL.EGTS_SR_TRACK_DATA (10 items, gap 5 items , 10 items) ...
PASSED
ECALL.EGTS_SR_TRACK_DATA (all items) ... PASSED
AUTH.EGTS_SR_TERM_IDENTITY+VEHICLE_DATA(simple) ... PASSED
```

Tests are passed successfully if there are no error messages in the program output and the output ends with the message

```
erorr count: 0
autotests PASSED
```

6.5 Testing of the source code of the service "Emergency Accident Response" in the part of an emergency SMS channel

Separate testing is not performed. Transport Level of the data transmission via SMS is tested in accord with the par. 6.2. The source code of the Application (Service) Level is similar to the corresponding code of the packaged data transmission (par. 6.4)

6.6 Testing of the source code in the part of control and data communication via SMS

The source code of the Application (Service) Level is similar to the corresponding code of the packaged data transmission (par. 6.3)

6.7. Testing of the source code in the part of testing and diagnostics of AT

Testing is performed the same way as in paragraph 6.1. In every series of tests packages of the Testing and Diagnostics of AT level are transmitted (service COMMANDS).

```
COMMANDS.EGTS_SR_COMMAND_DATA (simple COMCONF) ... PASSED
COMMANDS.EGTS_SR_COMMAND_DATA (COMCONF, max length) ... PASSED
COMMANDS.EGTS_SR_COMMAND_DATA (simple COM) ... PASSED
```

```
COMMANDS.EGTS_SR_COMMAND_DATA (COM, max length) ... PASSED
```

Tests are passed successfully if there are no error messages in the program output and the output ends with the message

```
erorr count: 0
autotests PASSED
```

6.8. Testing of the source code in the part of downloading the software

Testing is performed the same way as in paragraph 6.1. In every series of tests packages of the Level of Downloading the software (service FIRMWARE) are transmitted.

```
FIRMWARE.EGTS_SR_SERVICE_PART_DATA (simple, 64b, first) ... PASSED
FIRMWARE.EGTS_SR_SERVICE_PART_DATA (simple, 64b, second) ... PASSED
FIRMWARE.EGTS_SR_SERVICE_PART_DATA (simple, max length, first) ...
PASSED
FIRMWARE.EGTS_SR_SERVICE_PART_DATA (simple, max length, second) ...
PASSED
FIRMWARE.EGTS_SR_SERVICE_FULL_DATA (simple, 64b) ... PASSED
FIRMWARE.EGTS_SR_SERVICE_FULL_DATA (simple, max length) ... PASSED
```

Tests are passed successfully if there are no error messages in the program output and the output ends with the message

```
erorr count: 0
autotests PASSED
```

6.9. Requirements to the reliability of technical means and software

Correspondence to the MISRA-2004 standard is proved with the help of PC Lint (http://www.gimpel.com/) tool. Tests are performed by launching the packaged batch file chk.bat under the Windows command shell. During the tests all the source files that are included in the reference implementation (for the exception of source files of the tests) are analyzed consequently. For the settings of the chk.bat file and the PC Lint tool see “Source Code Deployment Instructions”.

The output of PC Lint should not contain messages that the text of the source files do not correspond to the MISRA-2004 standard.

6.10 Requirements for the source code to correspond to the standard ISO/IEC 9899:1999

Correspondence to the ISO/IEC 9899:1999 standard is proved in the process of compiling the programs with the GCC compiler in accord with the “Source Code Deployment Instructions”. The -std=iso9899:1990 parameter checks if the code corresponds to the standard at the assembly work. Absence of warnings from the compiler at the assembly work is a criterion of correspondence to the standard.

6.11 Testing of the source code of the service "Configuration of equipment and Service control" using test vectors and validation tools.

Stage 1. Tests are performed by calls the probe2.exe test program with a specified command line arguments. Test vectors placed in tvec_bin folder. The file name of test vector corresponds to charter in "Test vector set for validating reference implementation".

```
egts_transport_probe2.exe -i tvec_bin\5.2.1.bin
egts_transport_probe2.exe -i tvec_bin\5.2.2.bin
egts_transport_probe2.exe -i tvec_bin\5.2.3.bin
```

Below is given an example of program output for some packet:

```
>>packet received:
Protocol Version     PRV: 01h
Security Key ID     SKID: 00h
Prefix               PRF: 00h
Route                RTE: 0
Encryption Algorithm ENA: 00h
Compressed           CMP: 0
Priority              PR: 3
Header Length         HL: 11
Header Encoding       HE: 00h
Frame Data Length    FDL: 30
Packet Identifier    PID: 2
Packet Type           PT: 0001h
Peer Address         PRA: -
Recipient Address    RCA: -
Time To Live         TTL: -
Header Check Sum     HCS: B8h
-------------------------
 PT_APPDATA
-------------------------
RECORD:
Record Length           RL: 19
Record Number           RN: 1
 * Source Service On Device          SSOD: 0
 * Recipient Service On Device       RSOD: 1
 * Group                              GRP: 0
 * Record Processing Priority         RPP: 3
 * Time Field Exists                 TMFE: 0
 * Event ID Field  Exists            EVFE: 0
 * Object ID  Field Exists           OBFE: 1
Object Identifier      OID: 1
Event Identifier      EVID: -
Time                    TM: -
Source Service Type    SST: 4
Recipient Service Type RST: 4
SUBRECORD:
SR_COMMAND_DATA:
  Command Type               CT: 5 (COM)
  Command Confirmation Type  CCT: 0 (OK) (unexpected)
  Command Identifier         CID: 1 (00000001h)
  Source Identifier          SID: 1 (00000001h)
  FLAGS: ACFE= 0, CHSFE= 0
  Charset                    CHS: ---
  Authorization Code Length  ACL: ---
  Authorization Code         AC: ---
    COMMAND:
```

```
    Address                    ADR: 1 (0001h)
    Size                        SZ: 0
    Action                     ACT: 0 (00h)
    Command Code               CCD: 1 (0001h)
                                DT: size=1
\x01
>>
```

The test is successful if member-wise comparison with validation tool output is matched for each test vector.

Stage 2. Tests are performed by calls the probe2.exe test program with a specified command line arguments. Test program generates EGTS packet into binary file and textual description of packet content.

```
probe2.exe –n 13 –d 13.dump –x 13.bin
probe2.exe –n 14 –d 14.dump –x 14.bin
probe2.exe –n 15 –d 15.dump –x 15.bin
probe2.exe –n 16 –d 16.dump –x 16.bin
```

File <N>.bin contains the binary packet to pass to validation tools. The file <N>.dump contains the textual description of packet. The test is successful if member-wise comparison with validation tool output is matched for each dump generated.

6.12 Testing of the source code of the service "Emergency Accident Response" in the part of transmitting the acceleration pattern in case of a road accident, using test vectors and validation tools.

Stage 1. Tests are performed by calls the probe2.exe test program with a specified command line arguments. Test vectors placed in tvec_bin folder. The file name of test vector corresponds to charter in "Test vector set for validating reference implementation".

```
probe2.exe -i tvec_bin\5.1.1.bin
probe2.exe -i tvec_bin\5.1.2.bin
probe2.exe -i tvec_bin\5.1.3.bin
probe2.exe -i tvec_bin\5.1.4.bin
probe2.exe -i tvec_bin\5.1.5.bin
probe2.exe -i tvec_bin\5.1.6.bin
probe2.exe -i tvec_bin\5.1.7.bin
probe2.exe -i tvec_bin\5.1.8.bin
probe2.exe -i tvec_bin\5.1.9.bin
```

Below is given an example of program output for some packet:

```
>>packet received:
Protocol Version     PRV: 01h
Security Key ID     SKID: 00h
Prefix               PRF: 00h
Route                RTE: 0
Encryption Algorithm ENA: 00h
Compressed           CMP: 0
Priority              PR: 1
Header Length         HL: 11
Header Encoding       HE: 00h
Frame Data Length    FDL: 39
Packet Identifier    PID: 0
Packet Type           PT: 0001h
```

```
Peer Address          PRA: -
Recipient Address     RCA: -
Time To Live          TTL: -
Header Check Sum      HCS: CFh
-------------------------
 PT_APPDATA
-------------------------
RECORD:
Record Length           RL: 28
Record Number           RN: 0
 * Source Service On Device           SSOD: 0
 * Recipient Service On Device        RSOD: 1
 * Group                               GRP: 0
 * Record Processing Priority          RPP: 1
 * Time Field Exists                  TMFE: 0
 * Event ID Field  Exists             EVFE: 0
 * Object ID  Field Exists            OBFE: 1
Object Identifier      OID: 1
Event Identifier      EVID: -
Time                    TM: -
Source Service Type    SST: 4
Recipient Service Type RST: 4
SUBRECORD:
SR_COMMAND_DATA:
  Command Type               CT: 5 (COM)
  Command Confirmation Type  CCT: 0 (OK) (unexpected)
  Command Identifier         CID: 0 (00000000h)
  Source Identifier          SID: 0 (00000000h)
  FLAGS: ACFE= 1, CHSFE= 0
  Charset                    CHS: ---
  Authorization Code Length  ACL: 8 (08h)
  Authorization Code         AC:
testcode
    COMMAND:
    Address                   ADR: 1 (0001h)
    Size                       SZ: 0
    Action                    ACT: 0 (00h)
    Command Code              CCD: 274 (0112h)
                               DT: size=1
\x01
>>
```

The test is successful if member-wise comparison with validation tool output is matched for each test vector.

Stage 2. Tests are performed by calls the probe2.exe test program with a specified command line arguments. Test program generates EGTS packet into binary file and textual description of packet content.

```
probe2.exe –n 23 –d 23.dump –x 23.bin
probe2.exe –n 24 –d 24.dump –x 24.bin
probe2.exe –n 25 –d 25.dump –x 25.bin
probe2.exe –n 26 –d 26.dump –x 26.bin
probe2.exe –n 27 –d 27.dump –x 27.bin
```

File <N>.bin contains the binary packet to pass to validation tools. The file <N>.dump contains the textual description of packet. The test is successful if member-wise comparison with validation tool output is matched for each dump generated.

6.13 Testing the source code in the part of testing and diagnostics

of Vehicle Terminal, using test vectors and validation tools.

This test performed within the test 6.12.

6.14 Testing the source code in the part of downloading the software using test vectors and validation tools.

Stage 1. Tests are performed by calls the probe2.exe test program with a specified command line arguments. Test vectors placed in tvec_bin folder. The file name of test vector corresponds to charter in “Test vector set for validating reference implementation”.

```
probe2.exe -i tvec_bin\5.3.1.bin
probe2.exe -i tvec_bin\5.3.2.bin
```

Below is given an example of program output for some packet:

```
>>packet received:
Protocol Version     PRV: 01h
Security Key ID     SKID: 00h
Prefix               PRF: 00h
Route                RTE: 0
Encryption Algorithm ENA: 00h
Compressed           CMP: 0
Priority              PR: 1
Header Length         HL: 11
Header Encoding       HE: 00h
Frame Data Length    FDL: 99
Packet Identifier    PID: 1
Packet Type           PT: 0001h
Peer Address         PRA: -
Recipient Address    RCA: -
Time To Live         TTL: -
Header Check Sum     HCS: 57h
-------------------------
 PT_APPDATA
-------------------------
RECORD:
Record Length           RL: 88
Record Number           RN: 1
 * Source Service On Device          SSOD: 0
 * Recipient Service On Device       RSOD: 1
 * Group                              GRP: 0
 * Record Processing Priority         RPP: 1
 * Time Field Exists                 TMFE: 0
 * Event ID Field  Exists            EVFE: 0
 * Object ID  Field Exists           OBFE: 1
Object Identifier      OID: 1
Event Identifier      EVID: -
Time                    TM: -
Source Service Type    SST: 9
Recipient Service Type RST: 9
SUBRECORD:
SR_FULL_DATA_header:
SR_DATA_header:
  Object Type                     OT: 1 (CONFIG)
  Module Type                     MT: 1 (MAIN)
  Component or Module Identifier  CMI: 1 (01h)
  Version                         VER: 0 (0000h)
  Whole Object Signature          WOS: 5486 (156Eh)
  File Name                       FN: --
```

```
  Object Data              OD: len=78
0203="internet.beeline.ru,beeline,beeline"\x0D\x0A0204="95.255.255.25
5:8012"
\x0D\x0A0404=1
>>
```

The test is successful if member-wise comparison with validation tool output is matched for each test vector.

Stage 2. This test skipped because validation tool does not supports packets for software downloading.