Signaling No.7 Scenario Constructing, Analysis, Log-booking and Execution Suit

Version 0.3.1

Creating Message Templates and Operations Definitions

Sofia, 2013

This document and the information contained in it may not be published, distributed or reproduced without written permission of the author. Its purpose is to help readers to explore and acquire the 7-Scales project without any warranty regarding discrepancies with the other documents and the source code it refers to.

Table of Content

Document history

Date	Authors	Doc Rev.	SW Rel.	Subject/Reason for change	Status
2012-11	Ivelin Atanasov	1	0.2.2	Initial version	Complete
2013-03	Ivelin Atanasov	2	0.3.1	New SW release	Complete

1 Definitions

The 7-Scales project defines the following notions, related to messages, operations and information elements:

Message structure – a C-language structure that models an SS7 message that implements TCAP protocol. At MTP3 Level this is MSU (message signal unit); at SCCP level this is UDT or XUDT message. TCAP message contains operations of an Application Part like MAP, INAP, CAP.

Long context is a structure that models an Operation. It is implemented as a 2-dimentional array, each row of which is related to an I.E. that is present in the Operation and consists of consecutive Tags of the I.E.s that embrace it. The idea behind the Long context is very simple: you can get to any I.E. in a TCAP message by starting from the tag of the component (i.e. Invoke) and going down along the path of the tags of the consecutive I.E.s that embrace it until you step on the same tag as one of the I.E. that is your target. Here is an example:

For the input file input_msu.txt (MAP operation ProvideRoamingNumber[1]):

```
83 XX XX XX XX 09 80 03 0e 19 0b 12 07 00 12 04 53 89 XX XX XX XX 0b 12 06 00 12 04 53 89 XX XX XX XX 67 62 65 48 04 98 00 01
ef 6b 1e 28 1c 06 07 00 11 86 05 01 01 01 a0 11 60 0f 80 02 07 80 a1 09 06 07 04 00 00 01 00 03 03 6c 80 a1 39 02 01 01 02 01
04 30 31 80 08 82 04 05 00 70 12 06 f9 81 07 91 53 89 XX XX XX XX a5 08 0a 01 01 04 03 04 01 a0 88 07 91 53 89 XX XX XX XX 89
05 bc 96 d2 00 02 8f 02 05 c0 00 00 00
```

the Long context is:

```
//Operation data processing from file <input_msu.txt>

//Long context of DialoguePortion:
00006B 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
00006B 000028 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
00006B 000028 000006 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
00006B 000028 0000A0 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
00006B 000028 0000A0 000060 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
00006B 000028 0000A0 000060 000080 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
00006B 000028 0000A0 000060 0000A1 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
00006B 000028 0000A0 000060 0000A1 000006 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000

//Long context for Component 0:
0000A1 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000002 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000002 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000080 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000081 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 0000A5 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 0000A5 00000A 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 0000A5 000004 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000088 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000089 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 00008F 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
```

Each of the numbers there is a tag of an Operation. Each row gives a path from the Invoke tag (A1) to the I.E. the row refers to. Reference is very simple: 1st row corresponds to Invoke tag, 2nd – to the next I.E., InvokeID (02), etc.

For the time being the 7-Scales is capable to model Operations that have up to 16 levels of inclusion. This capability is controlled by a named constant, `MAX_DEPTH`, located in `tcapbase.h`.

Compare the above table with the log produced by the tool `dmtcap.exe` on the same message (the first decomposition is the Dialogue portion):

[1] Network sensitive information is replaced by 'XX' or 'nn'

```
Opening source file <input_msu.txt> for reading raw message ... success.

Input data:
83 XX XX XX XX 09 80 03 0e 19 0b 12 07 00 12 04 53 89 XX XX XX XX 0b 12 06 ...

After converting to Hex (OP, 139 bytes):
83 XX XX XX XX 09 80 03 0E 19 0B 12 07 00 12 04 53 89 XX XX XX XX 0B 12 06 ...

Writing results ...

00006B APPL (C) [      11] -- LL=1, LV=30 [i_bgn=43, i_end=137]
.000028 UNIV (C) [       8] -- LL=1, LV=28 [i_bgn=45, i_end=74]
..000006 UNIV (P) [       6] -- LL=1, LV=7 [i_bgn=47, i_end=74]
..= 00 11 86 05 01 01 01
..0000A0 CONT (C) [       0] -- LL=1, LV=17 [i_bgn=56, i_end=74]
...000060 APPL (C) [       0] -- LL=1, LV=15 [i_bgn=58, i_end=74]
....000080 CONT (P) [       0] -- LL=1, LV=2 [i_bgn=60, i_end=74]
....= 07 80
....0000A1 CONT (C) [       1] -- LL=1, LV=9 [i_bgn=64, i_end=74]
.....000006 UNIV (P) [       6] -- LL=1, LV=7 [i_bgn=66, i_end=74]
.....= 04 00 00 01 00 03 03
tcm.dp.ie_n=8, tcm.dp.ie_m=16

Operation parms decomposition:

0000A1 CONT (C) [       1] -- LL=1, LV=57 [i_bgn=77, i_end=135]
.000002 UNIV (P) [       2] -- LL=1, LV=1 [i_bgn=79, i_end=135]
.= 01
.000002 UNIV (P) [       2] -- LL=1, LV=1 [i_bgn=82, i_end=135]
.= 04
.000030 UNIV (C) [      16] -- LL=1, LV=49 [i_bgn=85, i_end=135]
..000080 CONT (P) [       0] -- LL=1, LV=8 [i_bgn=87, i_end=135]
..= 82 04 05 00 70 12 06 F9
..000081 CONT (P) [       1] -- LL=1, LV=7 [i_bgn=97, i_end=135]
..= 91 53 89 XX XX XX XX
..0000A5 CONT (C) [       5] -- LL=1, LV=8 [i_bgn=106, i_end=135]
...00000A UNIV (P) [      10] -- LL=1, LV=1 [i_bgn=108, i_end=115]
...= 01
...000004 UNIV (P) [       4] -- LL=1, LV=3 [i_bgn=111, i_end=115]
...= 04 01 A0
..000088 CONT (P) [       8] -- LL=1, LV=7 [i_bgn=116, i_end=135]
..= 91 53 89 XX XX XX XX
..000089 CONT (P) [       9] -- LL=1, LV=5 [i_bgn=125, i_end=135]
..= BC 96 D2 00 02
..00008F CONT (P) [      15] -- LL=1, LV=2 [i_bgn=132, i_end=135]
..= 05 C0
tcm.cm[0].ie_n=0, tcm.cm[12].ie_m=20

Resulting Operation's long context is saved in <frmlong.txt>

Control Operation's short context is saved in <dmtcap_00504DFC31.log>

Return code: 0
```

Note that most I.E.s are unique, but some may have multiple presence in an Operation. To access them there is another mechanism that is available at runtime.

Optimized context is another structure that models an Operation. As with the Long context, it is implemented as a 2-dimential array, each raw of which is related to an I.E. present in the Operation. It is possible to convert Long context in Optimized and vice versa. The idea behind the Optimized context is even simpler – it contains the same information in more efficient way both in terms of space and processing time. It is called sometimes "Short context". Here is the optimized context that corresponds to the long one (it is produced by `Gentco.exe`):

```
//[Gentco v2.0]: LongContext processing from file <ctx_lngMAPv3_PRNRQ.txt>

//[Gentcl v1.2]: Operation data processing from file <inputMAPv3_PRNRQ.txt>

//MAPv3 operation ProvideRoamingNumber

//OptimizedContext (Static thisIE structure) for operation 'idx_opProvideRoamingNumber_v3':

int tab_opProvideRoamingNumber_v3[12][7] = {
//    TAG          Dp   Up   Rt SCOPE  UNAME SPARE
{0x0000A1,         0,   0,   0,  11,     0,    0}, //  0
{0x000002,         1,   0,   2,   0,     0,    0}, //  1
{0x000002,         1,   0,   3,   0,     0,    0}, //  2
{0x000030,         1,   0,   0,   8,     0,    0}, //  3
{0x000080,         2,   3,   5,   0,     0,    0}, //  4
{0x000081,         2,   3,   6,   0,     0,    0}, //  5
{0x0000A5,         2,   3,   9,   2,     0,    0}, //  6
{0x00000A,         3,   6,   8,   0,     0,    0}, //  7
{0x000004,         3,   6,   0,   0,     0,    0}, //  8
{0x000088,         2,   3,  10,   0,     0,    0}, //  9
{0x000089,         2,   3,  11,   0,     0,    0}, // 10
{0x00008F,         2,   3,   0,   0,     0,    0}  // 11
};
```

As can be seen from the printout, it is called also Static thisIE structure of the Operation.

The term Optimized context is applied also to the Dynamic thisIE structure an example of which is presented on the following figure (produced by `dmtcap.exe` on the same message):

```
//dmtcap: Control data processing

#define MSG_LEN 139
unsigned char RAW_MSG[MSG_LEN] = {
0x83, 0xNN, 0xNN, 0xNN, 0xNN, 0x09, 0x80, 0x03, 0x0E, 0x19, 0x0B, 0x12, 0x07, 0x ...
};

//Runtime structure for Dialogue Portion:
#define DLOG_LEN 8
int DLOG_TAB[8][8] = {
//    Th    Dt   Dp   Up   Rt   Ll   Lv      Rc
{    43, 0x60,   0,   0,   0,   1,  30,      0}  , //   0
{    45, 0x20,   1,   0,   0,   1,  28,      0}  , //   1
{    47, 0x00,   2,   1,   3,   1,   7,      0}  , //   2
{    56, 0xa0,   2,   1,   0,   1,  17,      0}  , //   3
{    58, 0x60,   3,   3,   0,   1,  15,      0}  , //   4
{    60, 0x80,   4,   4,   6,   1,   2,      0}  , //   5
{    64, 0xa0,   4,   4,   0,   1,   9,      0}  , //   6
{    66, 0x00,   5,   6,   0,   1,   7,      0}    //   7
};

//Runtime structure for Component 0:
#define CM0_LEN 12
int CM0_TAB[CM0_LEN][8] = {
//    Th    Dt   Dp   Up   Rt   Ll   Lv      Rc
{    77, 0xa0,   0,   0,   0,   1,  57,      0}  , //   0
{    79, 0x00,   1,   0,   2,   1,   1,      0}  , //   1
{    82, 0x00,   1,   0,   3,   1,   1,      0}  , //   2
{    85, 0x20,   1,   0,   0,   1,  49,      0}  , //   3
{    87, 0x80,   2,   3,   5,   1,   8,      0}  , //   4
{    97, 0x80,   2,   3,   6,   1,   7,      0}  , //   5
{   106, 0xa0,   2,   3,   9,   1,   8,      0}  , //   6
{   108, 0x00,   3,   6,   8,   1,   1,      0}  , //   7
{   111, 0x00,   3,   6,   0,   1,   3,      0}  , //   8
{   116, 0x80,   2,   3,  10,   1,   7,      0}  , //   9
{   125, 0x80,   2,   3,  11,   1,   5,      0}  , //  10
{   132, 0x80,   2,   3,   0,   1,   2,      0}    //  11
};
```

Static thisIE structure is a structure of type Optimized context that is used to model the complete structure of an Operation.

Dynamic thisIE structure is also a structure of type Optimized context that is used to model an Operation at runtime.

Both structures match exactly each other in 2nd, 3rd and 4th columns – `DEPTH=Dt`, `OWNER=Up` and `RIGHT=Rt`. `OWNER` specifies which is the owner of this I.E., addressing the row of the table, at which the data related to the owner I.E. are located. `DEPTH` counts how many owners there are on the path up to the root. `RIGHT` addresses the next I.E. that is of the same level of depth and has the same owner as this I.E.

Considering that this I.E. is the 5th I.E. (the one `[Th]=97` and `[TAG]=000081`), the `[OWNER]=[Up]=3` as the 3rd I.E. directly encompasses this I.E., and `[RIGHT]=[Rt]=9` as the 9th I.E. is of the same level of depth and has the same owner as this I.E.

Static and dynamic thisIEs differ in the following. Static thisIE contains Tags while dynamic thisIE contain an index to the same Tag in the message; static thisIE contains holders for the Universal names of I.E.s (`UNAME`) and for indexes to functions that provide further treatment of these I.E.s (`SPARE`), while dynamic thisIE contain holders for the length of the Length of I.E.s (`Ll`), length of the Value of I.E.s (`Lv`), return code from processing exactly this I.E (`Rc`) and other details (`Dt`).

2 Creating operations definitions

Operations definitions are exactly static thisIE structures for the operations. This is a 2-step procedure – at first we produce the long context and then from the long context we produce the optimized context. On the first step we use `Gentcl.exe`, on the second step we use `Gentco.exe`.

2.1 Creating long context

`Gentcl.exe` is used as follows:

```
Gentcl –i <input_file.txt> -o <output_file.txt>
```

The input file has to be of the following shape:

```
//CS1P operation initialDP based on Personetta SCP
idx_opInitialDP_v0
len_opInitialDP_v0
tab_opInitialDP_v0
a1 30 02 01 00 02 01 00 30 28 80 01 4e 82 07 02 90 80 59 94 38 83 83 07 83 13 98 48 00 15 02 8a 0a 84 93 53 89 29 07 10 01 48 02 9a 02 60 01
```

The first line is considered to be comment with or without the '//'. The comment will be copied in the output file.

The following three lines are clear in their meaning. The index and the table are taken from the header file `cs1pbase.h` for CS1P and from `map_base.h` for MAP. The length is not a predefined name, but goes well with the others. The last line is the operation itself starting from the Invoke I.E. (Tag A1). It should finish with Enter and nothing on the next line.

The output is:

```
//[Gentcl v1.0.2]: Operation data processing from file 'inputCS1P_IDP.txt'
//CS1P operation initialDP based on Personetta SCP
//Index:
idx_opInitialDP_v0
//Length:
len_opInitialDP_v0
//Table:
tab_opInitialDP_v0
//Long context of input data:
0000A1 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000002 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000002 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000080 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000082 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000083 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 00008A 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 00009A 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 00009C 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
```

Here you can add the names of the I.E.s taken from the `cs1popdefs.h` for CS1P and `map_opdefs.h` for MAP.

```
//[Gentcl v1.0.2]: Operation data processing from file 'inputCS1P_IDP.txt'
//CS1P operation initialDP based on Personetta SCP
//Index:
idx_opInitialDP_v0
//Length:
len_opInitialDP_v0
//Table:
tab_opInitialDP_v0
//Long context of input data:
0000A1 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000002 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000002 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
0000A1 000030 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 -- operationArg
0000A1 000030 000080 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 -- serviceKey
0000A1 000030 000082 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 -- calledPartyNumber
0000A1 000030 000083 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 -- callingPartyNumber
0000A1 000030 00008A 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 -- locationNumber
0000A1 000030 00009A 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 -- forwardCallIndicators
0000A1 000030 00009C 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 -- eventTypeBCSM
```

You can add more rows either manually or from other outputs for the same operation. At this moment you are ready to supply the input for the Gentco.exe to produce the final result.

2.2 Creating optimized context

`Gentco.exe` is used as follows:

```
Gentco –i <input_file.txt> -o <output_file.txt>
```

In the course of the InitialDP example the result will be as follows:

```
//[Gentco v2.0]: LongContext processing from file <lngctxCS1P_IDP.txt>

//[Gentcl v1.0.2]: Operation data processing from file <inputCS1P_IDP.txt>

//CS1P operation initialDP based on Personetta SCP

//OptimizedContext (Static thisIE structure) for operation 'tab_opInitialDP_v0':

#define len_opInitialDP_v0 10

int tab_opInitialDP_v0[len_opInitialDP_v0][7] = {
//    TAG          Dp   Up   Rt SCOPE  UNAME SPARE
{0x0000A1,         0,   0,   0,   9,     0,    0}, //  0
{0x000002,         1,   0,   2,   0,     0,    0}, //  1
{0x000002,         1,   0,   3,   0,     0,    0}, //  2
{0x000030,         1,   0,   0,   6,     0,    0}, //  3 -- operationArg
{0x000080,         2,   3,   5,   0,     0,    0}, //  4 -- serviceKey
{0x000082,         2,   3,   6,   0,     0,    0}, //  5 -- calledPartyNumber
{0x000083,         2,   3,   7,   0,     0,    0}, //  6 -- callingPartyNumber
{0x00008A,         2,   3,   8,   0,     0,    0}, //  7 -- locationNumber
{0x00009A,         2,   3,   9,   0,     0,    0}, //  8 -- forwardCallIndicators
{0x00009C,         2,   3,   0,   0,     0,    0}  //  9 -- eventTypeBCSM
};

//Extended array for operation tab_opInitialDP_v0:
struct XArray xtab_opInitialDP_v0 = {
    len_opInitialDP_v0, /* length */
    (int*) tab_opInitialDP_v0 /* ptr to tab_opInitialDP_v0 */
};

//Now replace dummy record in tab_opXXX[] at position <idx_opInitialDP_v0> with the above XArray
```

Note that the index is used only in the comment in the end of the file. You have to copy the named constants that go in the comments onto the column `UNAME`. Then you place the array `tab_opInitialDP_v0` and the `XArray` object `xtab_opInitialDP_v0` in the header `cs1popdefs.h`. Finally, you replace the dummy row in the array `tab_opCS1P` [2]at position `idx_opInitialDP_v0` with the following:

```
{{len_opInitialDP_v0, (int*) tab_opInitialDP_v0}, {0, NULL}},
```

Make sure that all named constants for the I.E.s are defined in the header `cs1popdefs.h` and compile.

[2] Arrays `tab_opCS1P` and `tab_opMAP` are removed from `cs1popdefs.h` and `map_opdefs.h` since version v0-2-2.

3 Creating message templates

Message templates are created by the software tool `dmtcap2` for v0.3.1 onward. It is used as follows:

`dmtcap2 –a <SS7_ID> –c <Application_context>`

Here `SS7_ID` is taken from the header `tcapbase.h`: `UAP_ID, MAP1ID, MAP2ID, MAP3ID, CAP1ID, CAP2ID, CAP3ID, CAP4ID, INCS1ID, INCS2ID, INCS3ID`, and `INCS1PID`. Application_context is taken from `map_base.h`, `cap_base.h` and `cs1pbase.h`:

MAP	CAP	CS1P
acNetworkLocUp_v1	acCAP_gsmSSF_scfGeneric_v3	acCs1plus_ssp_to_scp_v0
acNetworkLocUp_v2	acCAP_gsmSSF_scfGeneric_v4	acCs1plus_assist_hoff_ssp_to_scp_v0
acNetworkLocUp_v3	acCAP_gsmSSF_scfAssistHandoff_v3	acCs1plus_ip_to_scp_v0
acLocationCancel_v1	acCAP_gsmSSF_scfAssistHandoff_v4	acCs1plus_scp_to_ssp_v0
acLocationCancel_v2	acCAP_scf_gsmSSFGeneric_v3	acCs1plus_scp_to_ssp_traffic_mngt_v0
acLocationCancel_v3	acCAP_scf_gsmSSFGeneric_v4	acCs1plus_scp_to_ssp_service_mngt_v0
acRoamingNbEnquiry_v1	ac_gsmSRF_gsmSCF_v3	acCs1plus_ssp_to_scp_service_mngt_v0
acRoamingNbEnquiry_v2	ac_gsmSRF_gsmSCF_v4	acCs1plus_data_mngt_v0
acRoamingNbEnquiry_v3	acCAP_gsmSSF_gsmSCF_v1	acCs1plus_scp_to_ssp_traffic_limit_v0
acIstAlerting_v3	acCAP_gsmSSF_gsmSCF_v2	
acLocInfoRetrieval_v1	acCAP_gprsSSF_gsmSCF_v3	
acLocInfoRetrieval_v2	acCAP_gprsSSF_gsmSCF_v4	
acLocInfoRetrieval_v3	acCAP_assist_gsmSSF_gsnSCF_v2	
acCallControlTransfer_v3	acCAP_gsmSCF_gprsSSF_v3	
acCallControlTransfer_v4	acCAP_gsmSCF_gprsSSF_v4	
acReporting_v3	acCAP_gsmSRF_gsmSCF_v2	
acCallCompletion_v3	ac_cap_sms_v3	
acServiceTermination_v3	ac_cap_sms_v4	
acReset_v1		
acReset_v2		
acHandoverControl_v1		
acHandoverControl_v2		
acHandoverControl_v3		
acEquipmentMngt_v1		
acEquipmentMngt_v2		
acEquipmentMngt_v3		
acInfoRetrieval_v1		

acInfoRetrieval_v2		
acInfoRetrieval_v3		
acInterVlrInfoRetrieval_v2		
acInterVlrInfoRetrieval_v3		
acSubscriberDataMngt_v1		
acSubscriberDataMngt_v2		
acSubscriberDataMngt_v3		
acTracing_v1		
acTracing_v2		
acTracing_v3		
acNetworkFunctionalSs_v1		
acNetworkFunctionalSs_v2		
acNetworkUnstructuredSs_v2		
acShortMsgGateway_v1		
acShortMsgGateway_v2		
acShortMsgGateway_v3		
acShortMsg_Relay_v1		
acShortMsgMO_Relay_v2		
acShortMsgMO_Relay_v3		
acSubscriberDataModifNotif_v3		
acShortMsgAlert_v1		
acShortMsgAlert_v2		
acMwdMngt_v1		
acMwdMngt_v2		
acMwdMngt_v3		
acShortMsgMT_Relay_v2		
acShortMsgMT_Relay_v3		
acImsiRetrieval_v2		
acMsPurging_v2		
acMsPurging_v3		
acSubscriberInfoEnquiry_v3		
acAnyTimeInfoEnquiry_v3		
acGroupCallControl_v3		
acGprsLocationUpdate_v3		
acGprsLocationInfoRetrieval_v3		
acGprsLocationInfoRetrieval_v4		
acFailureReport_v3		
acGprsNotify_v3		

acSs_InvocationNotification_v3		
acLocationSvcGateway_v3		
acLocationSvcEnquiry_v3		
acAuthenticationFailureReport_v3		
acShortMsgMT_RelayVGCS_v3		
acMm_EventReporting_v3		
acAnyTimeInfoHandling_v3		
acResourceManagement_v3		
acGroupCallInfoRetrieval_v3		

The input file has a fixed name, `input_msu.txt`. It should start at the MTP3 level (for national messages this is 0x83) and should consist of only one line, finishing with Enter. Here is an example with the InitialDP operation:

```
83, nn, nn, nn, nn, 09, 01, 03, 0E, 12, 0B, 52, FC, 00, 12, 04, 53, 89, nn, nn, nn, nn, 04, nn, nn, nn, nn, 7C, 62, 7A, 48, 04, 3E, 00, 00, C3, 6B, 35, 28,
33, 06, 07, 00, 11, 86, 05, 01, 01, 01, A0, 28, 60, 26, 80, 02, 07, 80, A1, 0E, 06, 0C, 2A, 86, 3A, 00, 89, 61, 33, 01, 01, 01, 00, 01, BE, 10, 28, 0E, FE,
0C, 30, 0A, 80, 01, 05, 81, 01, 05, 82, 02, 07, 39, 6C, 80, A1, 37, 02, 01, 00, 02, 01, 00, 30, 2F, 80, 01, 23, 82, 07, 02, 90, 80, 89, 04, 50, 31, 83, 07,
83, 13, 98, 44, 89, 08, 05, 8A, 0A, 84, 93, 53, 89, 29, 08, 10, 01, 48, 01, 9A, 02, 60, 01, BB, 05, 80, 03, 80, 90, A3, 9C, 01, 03, 00, 00
```

The resulting file is named as follows: `dmtcap_XXXXXXXXXX.log`. The content of this file for the InitialDP example is presented on the following figure.

Note that in the example below some of the octets are replaced by “nn” to hide network data that is not published usually.

```
//dmtcap: Control data processing

#define MSG_LEN 152
unsigned char RAW_MSG[MSG_LEN] = {
0x83, 0xnn, 0xnn, 0xnn, 0xnn, 0x09, 0x01, 0x03, 0x0E, 0x12, 0x0B, 0x52, 0xFC, 0x00,...
};

//Runtime structure for Dialogue Portion:
#define DLOG_LEN 15
int DLOG_TAB[15][8] = {
//   Th    Dt   Dp   Up   Rt   Ll   Lv      Rc
{    36, 0x60,   0,   0,   0,   1,  53,      0}  , //   0
{    38, 0x20,   1,   0,   0,   1,  51,      0}  , //   1
{    40, 0x00,   2,   1,   3,   1,   7,      0}  , //   2
{    49, 0xa0,   2,   1,   0,   1,  40,      0}  , //   3
{    51, 0x60,   3,   3,   0,   1,  38,      0}  , //   4
{    53, 0x80,   4,   4,   6,   1,   2,      0}  , //   5
{    57, 0xa0,   4,   4,   8,   1,  14,      0}  , //   6
{    59, 0x00,   5,   6,   0,   1,  12,      0}  , //   7
{    73, 0xa0,   4,   4,   0,   1,  16,      0}  , //   8
{    75, 0x20,   5,   8,   0,   1,  14,      0}  , //   9
{    77, 0xe0,   6,   9,   0,   1,  12,      0}  , //  10
{    79, 0x20,   7,  10,   0,   1,  10,      0}  , //  11
{    81, 0x80,   8,  11,  13,   1,   1,      0}  , //  12
{    84, 0x80,   8,  11,  14,   1,   1,      0}  , //  13
{    87, 0x80,   8,  11,   0,   1,   2,      0}    //  14
};

//Runtime structure for Component 0:
#define CM0_LEN 12
int CM0_TAB[CM0_LEN][8] = {
//   Th    Dt   Dp   Up   Rt   Ll   Lv      Rc
{    93, 0xa0,   0,   0,   0,   1,  55,      0}  , //   0
{    95, 0x00,   1,   0,   2,   1,   1,      0}  , //   1
{    98, 0x00,   1,   0,   3,   1,   1,      0}  , //   2
{   101, 0x20,   1,   0,   0,   1,  47,      0}  , //   3
{   103, 0x80,   2,   3,   5,   1,   1,      0}  , //   4
{   106, 0x80,   2,   3,   6,   1,   7,      0}  , //   5
{   115, 0x80,   2,   3,   7,   1,   7,      0}  , //   6
{   124, 0x80,   2,   3,   8,   1,  10,      0}  , //   7
{   136, 0x80,   2,   3,   9,   1,   2,      0}  , //   8
{   140, 0xa0,   2,   3,  11,   1,   5,      0}  , //   9
{   142, 0x80,   3,   9,   0,   1,   3,      0}  , //  10
{   147, 0x80,   2,   3,   0,   1,   1,      0}    //  11
};

//Runtime structure for message XXX:
struct TCAPmsu MSU_STRU = {
//MTP3 section - - - - - - - - - - - - - - - -
        MSG_LEN, /* int len */
        RAW_MSG, /* uchar* pbuf */
        0x83, /* unchar SIO, default: SCCP, National */
        1, /* int DPC */
        2, /* int OPC */
        3, /* int SLC */
        0x09, /* uchar H1H2, default: UDP */
//SCCP section - - - - - - - - - - - - - - - -
        0x01, /* uchar pcmh - protocol class & message handling */
        { /* Called_Pty */
                10, /* int GT_idx */
                11, /* int GT_len */
                0, /* int indSPC */
                0, /* int indSSN */
                0, /* int ind_GT */
                0, /* int indRtg */
                0, /* int SPC */
                0, /* int SSN */
                4, /* int NOAind, default: International */
                0, /* int TT_ind */
                1, /* int NP_ind, default: E.164 */
                0, /* int ES_ind */
                0, /* int AD_len */
                {0, } /* unsigned char AdrDig[20] */

        },
        { /* CallingPty */
                22, /* int GT_idx */
                4, /* int GT_len */
                0, /* int indSPC */
                0, /* int indSSN */
                0, /* int ind_GT */
                0, /* int indRtg */
                0, /* int SPC */
                0, /* int SSN */
                4, /* int NOAind, default: International */
                0, /* int TT_ind */
                1, /* int NP_ind, default: E.164 */
                0, /* int ES_ind */
                0, /* int AD_len */
                {0, } /* uchar AdrDig[0] */
        },
        0x00, /* uchar Hop Counter */
        { /* Segmentation - for XUDT and LUDT */
                0x00, /* uchar - control field */
                {0x00, 0x00, 0x00,} /* uchar[4] - local ref */
        },
        124, /* int ldta */
//TCAP section - - - - - - - - - - - - - - - -
        -1, /* int SESidx */
        28, /* int idta */
        0x62, /* uchar type */
        1, /* int llen */
        122, /* int vlen */
        { /* otid */
                30, /* int idx0 */
                1, /* int llen */
                4  /* int vlen */
        },
        { /* dtid */
                0, /* int idx0 */
                0, /* int llen */
```

```
        0  /* int vlen */
    },
    { /* struct TCAPdlog dp */
        0, /* int rcod */
        36, /* int idx0 */
        55, /* int len */
        DLOG_LEN, /* int ie_n */
        23, /* int ie_m */
        DLOG_TAB, /* int* ie_p */
        0x60, /* uchar type */
        9, /* int s7ap */
        0, /* int accod */
        1, /* int acidx */
        1, /* int acver */
        0, /* int acres */
        { /* protv */
            1, /* pvset */
            0x80, /* uchar bmask */
            1, /* n */
            56  /* int idx to bstr */
        }, /* protv */
        {0x00, 0x00, 0x08} /* struct <noname> inf */
    },
    { /* struct TCAPcptn cp */
        91, /* int idx0 */
        1, /* int llen */
        59,  /* int vlen */
        1  /* int cm_n */
    },
    { /* struct TCAPcomp cm[8] */
        { /* Component 0 */
            0, /* int rcod */
            93, /* ind idx0 */
            57, /* int len */
            CM0_LEN, /* int ie_n */
            20, /* int ie_m */
            CM0_TAB, /* int* ie_p */
            0xA1, /* uchar type */
            0x00, /* int invID */
            0x00, /* int lnkID */
            0, /* int opcod */
            8, /* int opidx */
            101, /* int iarg */
        },
    },
    0, /* uchar misc */
};
```

This output needs some manual modification to be ready for a message template. The modification consists of writing names of the named constants that are used here: `MSG_LEN, RAW_MSG, DLOG_LEN, DLOG_TAB, CM0_LEN, CM0_TAB, …, CM7_LEN, CM7_TAB`, and `MSU_STRU`. The following rules are applied so far:

- For the header file – `cs1pvV_acM_opNN.h`, where V=version, M=Application Context, NN=consecutive number of the header file, starting from 01
- For `MSG_LEN` - `CS1PACx_OPnnOBmm_LEN`, where x=Application Context, nn=consecutive number of the Operations Package (the number of the header file, nn=NN), mm=consecutive number of the Operations Bundle (a set of Operations put in a message)
- For `RAW_MSG` - `CS1PACx_OPnnOBmm_MSU`, where x=Application Context, nn=consecutive number of the Operations Package (the number of the header file, nn=NN), mm=consecutive number of the Operations Bundle (a set of Operations put in a message)
- `DLOG_TAB` is a named constant that contains NULL and has to be replaced if the message carries a Dialogue Portion. For the example it has the form of `CS1PDLOG_ACxRQ_TAB`, where n=Application Context.
- For `CM0_TAB` the name is `CS1PACx_OPnnOBmmYYY_TAB`, where x= Application Context, nn= Operations Package, mm= Operations Bundle and YYY is the short name of the Operation that is modeled by this structure (IDP in this case).
- Etc.

Note that an Operation can be used in many Application Contexts. It may be needed to change the Application Context of the template message that conveys the Operation at runtime.