Seagull - OpenCall TCAP

Table of contents

1. TCAP protocol details

The TCAP implementation in Seagull consist in the full implementation and support of HP OpenCall SS7 (http://www.hp.com/go/opencall/) TCAP API. This implementation supports both TCAP over SS7/E1/T1/J1/V35 and also TCAP over SIGTRAN / M3UA or SUA.

Both ITU and ANSI flavors are supported. A Seagull dictionary is available for each of those.

Warning:

For ANSI, Seagull only supports ANSI90 (HP OpenCall SS7 does not officially support ANSI96). It uses TCAP_API100 compile flag and must be linked to first version of the libSS7utilsAAA.so library (libSS7utilAAA.so.1) if several versions of this library exist on the system (refer to Generic configuration section).

2. Compiling Seagull with OCTCAP support

By default, OCTCAP support is disabled when compiling Seagull. To enable OCTCAP support, you must uncomment the "HP OC TCAP transport library" section in build.conf file and recompile Seagull.

In addition, OCTCAP include files must be present on the system on which Seagull is compiled. It is advised to compile Seagull on the OCSS7 platform.

3. Getting started with TCAP

3.1. First try

Note:

As HP OpenCall SS7 (http://www.hp.com/go/opencall/) is a pre-requisite for TCAP support in Seagull, an OCSS7 compliant platform must be selected.

So that you can get familiar with Seagull, here is an example that will launch one TCAP server (a server expects a message as the first scenario command) and one TCAP client (a client sends a message as the first scenario command). The scenario is the following:

Open two terminal sessions. Terminal 2 will be the server and Terminal 1 the client. Examples are located in the "run" directory. So the first thing you need to do is to go in this directory (in both terminal windows):

```
cd run
```

In Terminal 2 window type:

```
./start_server_itu.ksh
```

In Terminal 1 window type:

```
./start_client_itu.ksh
```

On Terminal 2 (server side), you will see:

```
|-------------------------+-----------------------------+-------------------------|
| Start/Current Time      |       2005-12-14 10:04:11 |     2005-12-14 10:06:53 |
|-------------------------+-----------------------------+-------------------------|
|    Counter Name         |      Periodic value         |     Cumulative value    |
|-------------------------+-----------------------------+-------------------------|
| Elapsed Time            | 00:00:01:008                | 00:02:41:596            |
| Call rate (/s)          |   75.397                    |   41.505                |
|-------------------------+-----------------------------+-------------------------|
| Incoming calls          |       76                    |     6707                |
| Outgoing calls          |        0                    |        0                |
| Msg Recv/s              |  149.802                    |   82.985                |
| Msg Sent/s              |  149.802                    |   82.979                |
| Unexpected msg          |        0                    |        0                |
| Current calls           |        3                    |    0.019                |
|-------------------------+-----------------------------+-------------------------|
| Successful calls        |       75                    |     6704                |
| Failed calls            |        0                    |        0                |
| Refused calls           |        0                    |        0                |
| Aborted calls           |        0                    |        0                |
| Timeout calls           |        0                    |        0                |
|-------------------------+-----------------------------+-------------------------|
| Last Info               | Incomming traffic                                     |
| Last Error              | No error                                              |
|--- Next screen : Press key 1 ----------------------- [h]: Display help ------|
```

You can take a look at the log files, which contain the TCAP messages exchanged. By default, those files are respectively `client.date.log` and `server.date.log`, suffixed with the date and time at which traffic started.

How easy was that? Now let's jump to the next section to learn how all that works.

3.2. First try explained

Here is the script (start_client.ksh) that launched the client in our example:

```
#!/bin/ksh

export LD_LIBRARY_PATH=/usr/local/bin

seagull -conf ../config/conf.client-itu.xml -dico ../config/octcap-itu-dictionnary.xml
-scen ../scenario/client-itu.xml -log ../logs/client.log -llevel ET
```

Note:

On some systems, you might need to include the path to the HP-OC SS7 api into the environment variable SHLIB_PATH : "/opt/OC/lib/" for SS7 3.x and "/opt/HP-AIN/SS7_WBB/sharedlib" for SS7 2.x . Add the following export in your Seagull script: "export SHLIB_PATH=$SHLIB_PATH:/opt/OC/lib/" for HP-OC SS7 3.x or "export SHLIB_PATH=$SHLIB_PATH:/opt/HP-AIN/SS7_WBB/sharedlib/" for HP-OC SS7 2.x .

Note:

On some HP-UX systems, you might need to include the following export in your Seagull script: "export SHLIB_PATH=/usr/local/bin".

This example is based on one client that sends the TC_BEGIN and exchanges TC_CONTINUE messages with one server that receives a TC_BEGIN and answers by TC_CONTINUE messages, until it sends the TC_END message.

Both sides are relying on the TCAP dictionary provided with the tool. We take the example of an ITU configuration. The dictionnary is: octcap-itu-dictionnary.xml Refer to dictionary configuration section for

more information on the format of this dictionary. The dictionary is specified using the `-dico` parameter on the command line.

The generic configuration (including network and other parameters) is different for the client and the server. The client uses `conf.client-itu.xml` and the server uses `conf.server-itu.xml`. The configuration file is specified using the `-conf` parameter on the command line.

Here are both files:

conf.client-itu.xml

```
<?xml version="1.0"
encoding="ISO-8859-1"?>
<configuration name="Simple TCAP
Client Conf">

  <define entity="transport"
    name="trans-octcap"
    file="libtrans_octcap.so"
create_function="create_ctransoctcap_
delete_function="delete_ctransoctcap_
    init-args="flavour=WBB">
  </define>

  <define entity="channel"
    name="channel-1"
    protocol="octcap-itu"
    transport="trans-octcap"
    open-args=
     "class=SS7_Stack_2;ossn=20;
      application=2;instance=2">
  </define>

  <define entity="traffic-param"
          name="call-rate"
          value="1">
  </define>

  <define entity="traffic-param"
          name="display-period"
          value="1">
  </define>

  <define entity="traffic-param"
          name="log-stat-period"
          value="1">
  </define>

  <define entity="traffic-param"
          name="log-stat-file"
value="../logs/client-stat.csv">
  </define>

  <define entity="traffic-param"
          name="call-timeout-ms"
          value="5000">
  </define>

  <define entity="traffic-param"
          name="max-send"
          value="1000">
  </define>

  <define entity="traffic-param"
          name="max-receive"
```

conf.server-itu.xml

```
<?xml version="1.0"
encoding="ISO-8859-1"?>
<configuration name="Simple TCAP
Server Conf">

  <define entity="transport"
    name="trans-octcap"
    file="libtrans_octcap.so"
create_function="create_ctransoctcap_instance"
delete_function="delete_ctransoctcap_instance"
    init-args="flavour=WBB">
  </define>

  <define entity="channel"
    name="channel-1"
    protocol="octcap-itu"
    transport="trans-octcap"
    open-args=
     "class=SS7_Stack_1;ossn=10;
      application=1;instance=1">
  </define>

  <define entity="traffic-param"
          name="display-period"
          value="1">
  </define>

  <define entity="traffic-param"
          name="log-stat-period"
          value="1">
  </define>

  <define entity="traffic-param"
          name="log-stat-file"
value="../logs/server-stat.csv">
  </define>

  <define entity="traffic-param"
          name="msg-check-level"
          value="P">
  </define>

  <define entity="traffic-param"
name="display-scenario-stat"
          value="false">
  </define>

  <define entity="traffic-param"
name="display-protocol-stat"
          value="true">
  </define>

  <define entity="traffic-param"
name="log-protocol-stat-period"
```

```
          value="1000">
  </define>

  <define entity="traffic-param"
name="max-simultaneous-calls"
          value="5000">
  </define>

  <define entity="traffic-param"
   name="external-data-file"
   value="external_data.csv">
  </define>

  <define entity="traffic-param"
   name="external-data-select"
   value="sequential">
  </define>

  <define entity="traffic-param"
          name="msg-check-level"
          value="P">
  </define>

  <define entity="traffic-param"
          name="msg-check-behaviour"
          value="E">
  </define>

  <define entity="config-param"
          name="dest-routing-type"
          value="DPC_SSN">
  </define>

  <define entity="config-param"
          name="orig-routing-type"
          value="DPC_SSN">
  </define>

  <define entity="config-param"
    name="orig-address-pc"
    value="2">
  </define>

  <define entity="config-param"
    name="dest-address-pc"
    value="1">
  </define>

  <define entity="config-param"
    name="orig-address-ssn"
    value="20">
  </define>

  <define entity="config-param"
    name="dest-address-ssn"
    value="10">
  </define>

</configuration>
```

```
          value="5">
  </define>

  <define entity="traffic-param"
name="log-protocol-stat-name"
          value="all">
  </define>

  <define entity="traffic-param"
name="log-protocol-stat-file"
value="../logs/server-protocol-stat.csv">
  </define>

  <define entity="traffic-param"
          name="max-send"
          value="1000">
  </define>

  <define entity="traffic-param"
          name="max-receive"
          value="1000">
  </define>

  <define entity="traffic-param"
name="max-simultaneous-calls"
          value="5000">
  </define>

  <define entity="traffic-param"
          name="call-timeout-ms"
          value="30000">
  </define>

  <define entity="config-param"
          name="orig-routing-type"
          value="DPC_SSN">
  </define>

  <define entity="config-param"
          name="dest-routing-type"
          value="DPC_SSN">
  </define>

  <define entity="config-param"
    name="orig-address-pc"
    value="1">
  </define>

  <define entity="config-param"
    name="dest-address-pc"
    value="2">
  </define>

  <define entity="config-param"
    name="orig-address-ssn"
    value="10">
  </define>

  <define entity="config-param"
    name="dest-address-ssn"
    value="20">
  </define>

</configuration>
```

Table 1: Example client and server configuration

Warning:

Do not forget to modify the classname and OSSN (in the open-args command) and the OPC, DPC, OSSN and DSSN (at the end of the files) for each SS7 configuration.

Note:

Refer to generic configuration section for more information on the format and possible values.

Now comes the real stuff: the scenario.

First, the scenario source: tcap.conf.client-itu.xml (tcap.conf.client.xml.html)

And now the commented version:

Scenario	Comments
<?xml version="1.0" encoding="ISO-8859-1" ?> <scenario> <counter> <counterdef name="client-uid-counter" init="1"></counterdef> </counter> <traffic> <send channel="channel-1"> <action> <set-value name="uid" format="$(client-uid-counter)"></set- <restore-from-external field="1" entity="TC_INVOKE" sub-entity="operation-data" begin="5" end="10"></restore-from-external> </action> <primitive name="TC_BEGIN" termination="BASIC"> <component name="TC_INVOKE" instance="InitialDP-data" class="2" timeout="60000" operation-code="0" invoke-id="1" operation-data="0x3016a00e820c48656c6c6f2c20776f726c </component> </primitive> <action> <inc-counter name="client-uid-counter"></inc-count <start-timer></start-timer> </action> </send> <receive channel="channel-1"> <primitive name="TC_CONTINUE" o-address-pc="$(dest-address-pc)" o-address-ssn="$(dest-address-ssn)" d-address-pc="$(orig-address-pc)"	XML header Counters definition In this section are initialized the counters that will then be used during the calls. For example here: we initialize the counter that will be incremented every time a call is started. Actions before sending the first TC_BEGIN message: set the uid field, as defined dictionary, with the value or the client-uid-counter. restore-from-external gets some data from an external file. Send the primitive TC_BEGIN with a TC_INVOKE component. The instance attribute identifies the component within the primitive, in case of multiple TC_INVOKE in the same primitive. The operation data is obtained via an external encoder. See chapter about external data management to know how some part of these operation data can be replaced by external data. Increment the uid-counter. Start the timer to measure response time End of send section for the TC_BEGIN.

```
d-address-ssn="$(orig-address-ssn)">
      <component name="TC_INVOKE"
instance="Client-1-data"
                 class="1"
                 invoke-id="1"
                 operation-code="18"
                 operation-data=
"0x3016a00e820c48656c6c6f2c20776f726c
      </component>

      <component name="TC_INVOKE"
instance="Client-2-data"
                 operation-code="23"
                 invoke-id="2"
                 operation-data=
"0x3016a00e820c48656c6c6f2c20776f726c
      </component>
    </primitive>

    <action>
     <check-value
name="d-address-pc"
behaviour="error">
     </check-value>
     <check-value
name="o-address-pc"
behaviour="error">
     </check-value>
     <check-value
name="d-address-ssn"
behaviour="error">
     </check-value>
     <check-value
name="o-address-ssn"
behaviour="error">
     </check-value>

      <store name="SERVER-UID"
entity="uid"></store>
      <store name="SERVER-PID"
entity="pid"></store>

    </action>
  </receive>

  <send channel="channel-1">
    <action>
      <restore name="SERVER-UID"
entity="uid"></restore>
      <restore name="SERVER-PID"
entity="pid"></restore>
    </action>
    <primitive name="TC_CONTINUE">
      <component name="TC_INVOKE"
instance="InitialDP-data"
                 class="1"
                 timeout="60000"
                 invoke-id="2"
                 operation-code="33"
operation-data="0x30818899005011">
      </component>
    </primitive>
  </send>

  <receive channel="channel-1">
    <primitive name="TC_END"
```

```
Receive the TC_CONTINUE primitive
sent by the server.
The values are set, they receive the
values of the
parameters in the configuration
file.
Originating and destination values
need to be inverted
as this is an incoming message, so
the values were set
by the remote: its DPC is us,
therefore, it is our OPC.
This primitive contains 2 TC_INVOKE
components.
The first one
(instance=Client-1-data).



The second one
(instance=Client-2-data).









Performs checks on the values of the
opc, dpc, ossn
and dssn. The comparison is done
between the values
received in the TC_CONTINUE and the
values expected
in the scenario.
The behaviour indicates the we end
in error in case
of wrong value. This would mean that
the message was
not aimed for us.



Those 2 values identifies the call.
The uid is incremented by Seagull
(it is taken from
the client-uid-counter initiated at
the beginning)
and the pid is given by the stack.
See the following
schema for more details.









Send a TC_CONTINUE primitive, with a
TC_INVOKE component.
```

```
o-address-pc="$(dest-address-pc)"
o-address-ssn="$(dest-address-ssn)"
d-address-pc="$(orig-address-pc)"
d-address-ssn="$(orig-address-ssn)">

      <component name="TC_RESULT_L"
                 invoke-id="2"
                 operation-code="33"
operation-data="0x310E9F82310A0203040
      </component>
    </primitive>
    <action>
      <stop-timer></stop-timer>
    </action>
  </receive>

</traffic>

<default>
  <receive channel="channel-1">
    <primitive
name="SCCP_USER_STATUS">
    </primitive>
  </receive>
</default>

<default behaviour="failed">
  <receive channel="channel-1">
   <primitive name="NO_PRIMITIVE">
      <component
name="TC_L_CANCEL"></component>
    </primitive>
  </receive>
</default>

<default behaviour="failed">
  <receive channel="channel-1">
    <primitive name="TC_P_ABORT">
    </primitive>
  </receive>
</default>
</scenario>
```

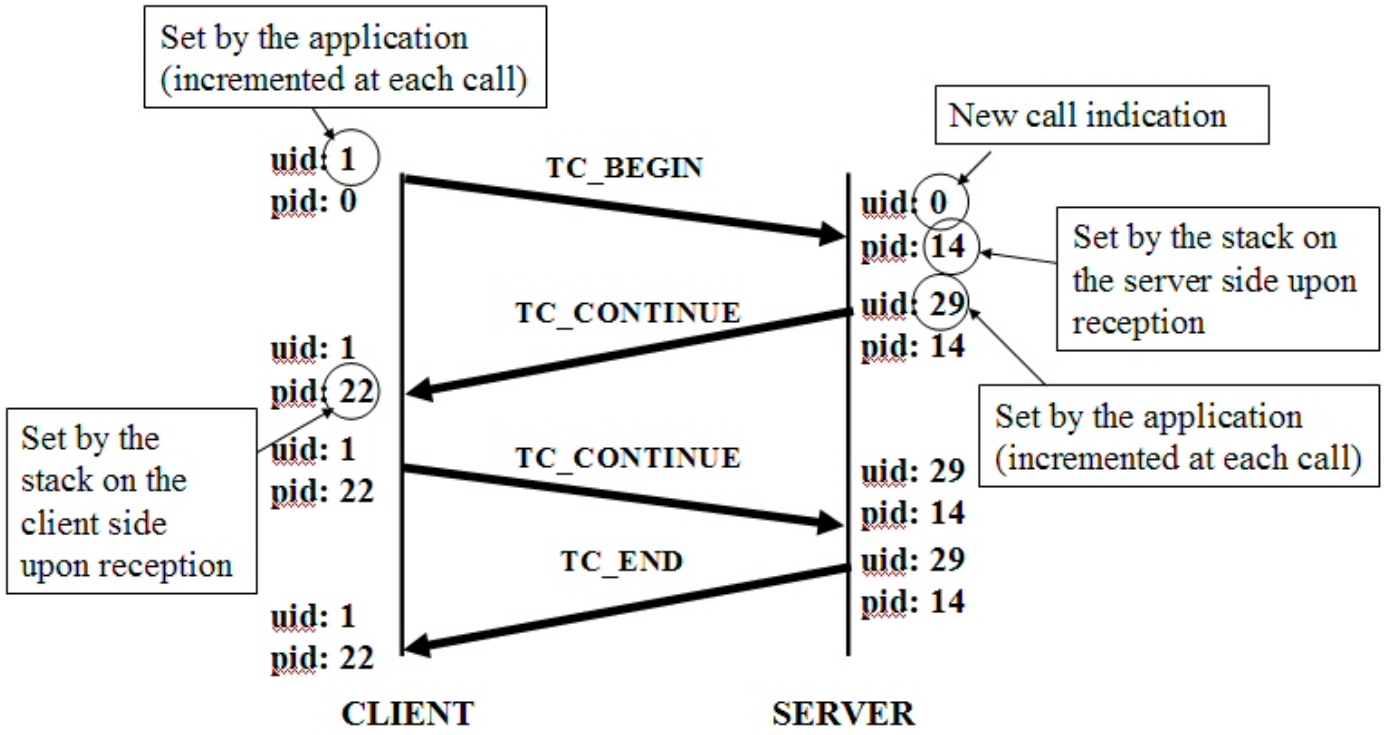
```
Receive the TC_END primitive



with a TC_RESULT_L component.




End the timer.


End of traffic description

One default scenario.
It describes the behaviour in case
of reception
of an unexpected SCCP_USER_STATUS
primitive.

The behaviour can be either failed
or success.
It is not mandatory to specify it,
by default,
it is success.
Another default scenario.
It describes the behaviour (here:
failed) in
case of reception of an unexpected
TC_L_CANCEL
primitive.



Another default scenario.
It describes the behaviour (here:
failed) in
case of reception of an unexpected
TC_P_ABORT
primitive.


End of scenario
```

Note:

A scenario can be considered as failed or success based on conception choices. In our example, a default scenario for the reception of an unexpected TC_L_CANCEL is considered as failed. But you could make the choice to consider it as successful, depending on your needs.

Note:

Here is an explanation on how are generated the uid and pid which are necessary to identify a call. As shown on the schema, the uid is set by the application ONCE for each call at the beginning. This is done using the "set-value" function (as shown in the example above). Note also that it is mandatory to restore the values in each message that you send, as done in the scenario above.



Note:
In this case, the application is Seagull and the stack is the OCSS7/TCAP (http://www.hp.com/go/opencall/) stack.

4. Using Seagull

4.1. Transport protocols and channels for TCAP

To send messages, you first have to define the transport that you use. This is done in the configuration file (see the example below). The transport is caracterized by a name and a file, which is the dynamic library used. This dynamic library is delivered with Seagull. It will call function from the correct library. To find the final library, it will use the parameters set in the init-args field. In **init-args**, you can specify the following parameters:

- flavour (possible values: WBB, WAA for ITU (sccp_service_kind is set to TCX_SCCP_SERVICE_ITU_WB) and AAA, ABB for ANSI (sccp_service_kind is set to TCX_SCCP_SERVICE_REGULAR))
- path to the OCSS7's libSS7util.sl shared library (not necessary for OCSS7 >= 3.x - /opt/HP-AIN/SS7_WBB/sharedlib for OCSS7 2.2)
- library (name of the libSS7util shared library) - libSS7utilFLAVOUR.sl by default (FLAVOUR being WBB, AAA, WAA or ABB).

Only the flavour parameter is mandatory for OCTCAP.

Some examples:

- init-args="flavour=AAA": for an OCSS7 3.x+ with AAA (ANSI) flavour
- init-args="path=/opt/HP-AIN/SS7_WBB/sharedlib;flavour=WBB;library=libSS7util.sl": for an OCSS7 2.2 with WBB (ITU) flavour

Warning:
For ANSI, with HP OpenCall SS7 3.2 on IA64 and with HP OpenCall 3.3 and upper on all plaform, the library name must be specified to "library=libSS7util.so.1".

Then you can open one channel for the transport that you have defined.

Note:
You can open only one TCAP channel !

A channel makes the link between a transport and a protocol.

When defining a channel, you specify the classname, OSSN, application id and instance id.

Note:
The application id needs to be unique for each application launched.

Those definitions are done in the configuration files, such as presented in those examples:

```
Example for ITU:
  <define entity="transport"
    name="trans-octcap"
    file="libtrans_octcap.so"
    create_function="create_ctransoctcap_instance"
    delete_function="delete_ctransoctcap_instance"
    init-args="flavour=WBB">
  </define>

  <define entity="channel"
    name="channel-1"
    protocol="octcap-itu"
    transport="trans-octcap"
    open-args="class=SS7_Stack_2;ossn=20;application=2;instance=2">
  </define>
```

4.2. TCAP specific parameters management

4.2.1. Invoke id

The invoke identifier is one of the header parameters of a component. It is assigned by the invoking side at invocation time. Each invoke id value is associated with an operation invocation. The management of this invoke id is done on the end that invokes the operation. Invoke id can be different on both sides. An invoke id value may be reallocated when the corresponding operation is done. The value of the invoke id has to be between 0 and 255. Here is an example of a client. This client sends a TC_INVOKE with invoke id 1. It then waits for 2 TC_INVOKE components within a primitive TC_CONTINUE:

```
  <send channel="channel-1">
    <primitive name="TC_BEGIN" termination="BASIC">
      <component name="TC_INVOKE" instance="InitialDP-data"
                 class="2"
```

```
                timeout="60000"
                operation-code="0"
                invoke-id="1"
                 operation-data=
      "0x3016a00e820c48656c6c6f2c20776f726c64810100820100">
      </component>
    </primitive>
  </send>

  ...

  <receive channel="channel-1">
    <primitive name="TC_CONTINUE">
      <component name="TC_INVOKE" instance="Client-1-data"
                class="1"
                invoke-id="1"
                operation-code="18"
                operation-data=
                  "0x3016a00e820c48656c6c6f2c20776f726c64810100820199">
      </component>

      <component name="TC_INVOKE" instance="Client-2-data"
                operation-code="23"
                invoke-id="2"
                operation-data=
                  "0x3016a00e820c48656c6c6f2c20776f726c64810100820120">
      </component>
    </primitive>

    </action>
  </receive>
```

4.2.2. Dialogue portion (ITU only)

In the header of the TCAP primitives, you find the dialogue portion. This dialogue portion is composed of:

- the application context name: it is the identifier of the application context,
- the user information: it is information exchanged between the TCAP users; this information is completely transparent to SS7 TCAP.

It is often useful to be able to extract those information from a received message to be able to inject them into a message to be sent. Here is an example of how to treat those information to be able to do so.

In this example, the client sends a message with the user informtion and aplication context name of the dialogue portion set. The server that receives the message extract some information from the user information to send it back, in a different order. It is important to change the order of the information as some fields of the user information refer to the origin and the destination of the message.

Extract from the client scenario:

```
<send channel="channel-1">
    <action>
       ...
       <restore-from-external field="3"
entity="dlg-user-information"></restore-from-external>
       <!-- first call, the dlg-user-information is
            28 1b 06 07 04000001010502 a010300e a0 04020269aa a1 060204010000ce
       -->
       ...
    </action>

    <primitive name="TC_BEGIN"
```

```
              dlg-app-ctxt-name="0x060704000001003203">
    ...
  </send>
```

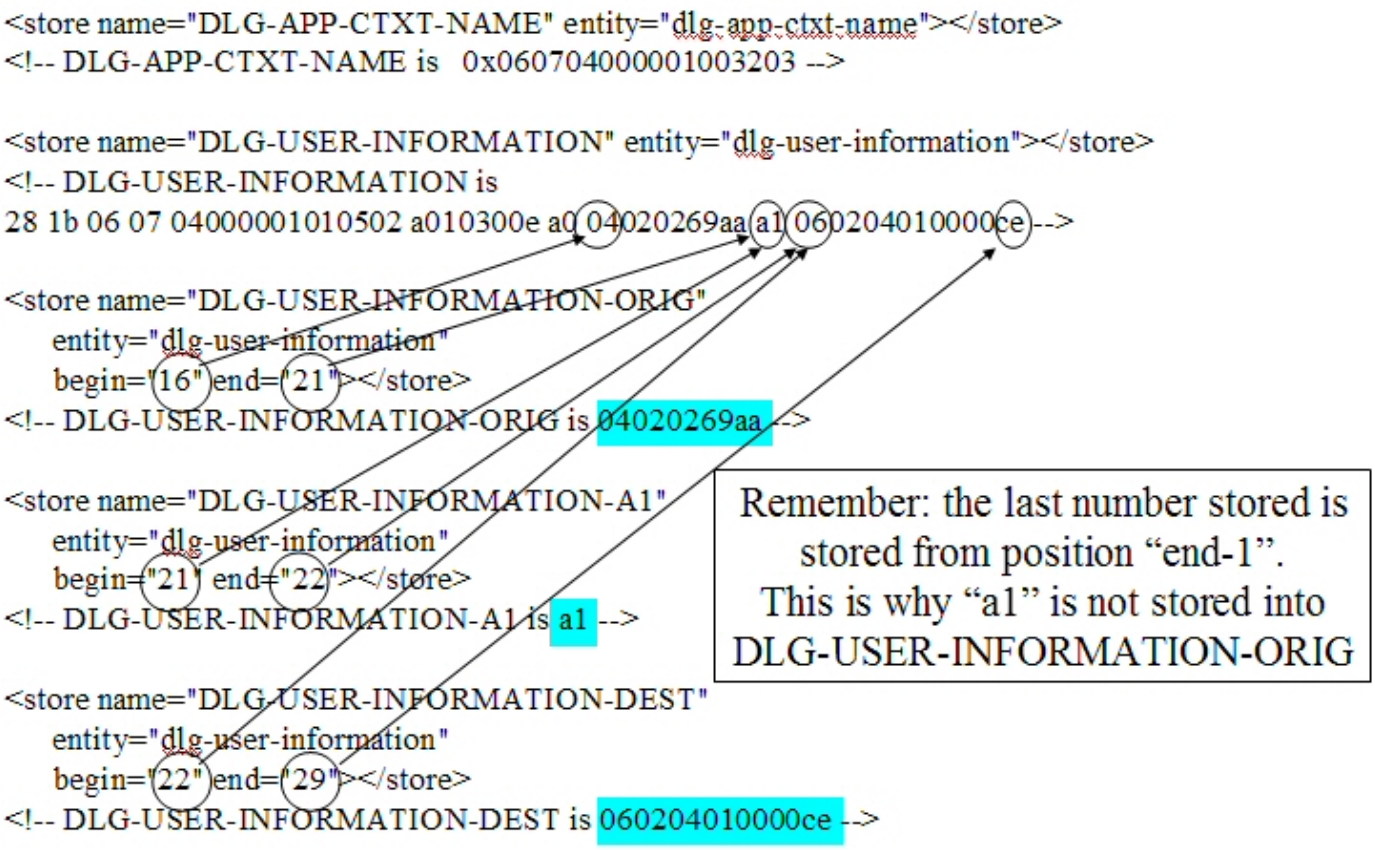
The external data file:

```
    "string"     ;  "string"                ; "number" ; "string";

# this is an example

    "0472826017" ;  "0x30343732413236303631" ; "10"     ;
"0x281b06070400000101050 2a010300ea004020269aaa1060204010000ce" ;
    "0472826027" ;  "0x30343732423236303632" ; "10"     ;
"0x281b06070400000101050 2a010300ea004020269aaa1060204010001ce" ;
    "0472826037" ;  "0x30343732433236303633" ; "10"     ;
"0x281b06070400000101050 2a010300ea004020269aaa1060204010002ce" ;
    "0472826047" ;  "0x30343732443236303634" ; "10"     ;
"0x281b06070400000101050 2a010300ea004020269aaa1060204010003ce" ;
    "0472826057" ;  "0x30343732453236303635" ; "10"     ;
"0x281b06070400000101050 2a010300ea004020269aaa1060204010004ce" ;
    "0472826067" ;  "0x30343732463236303636" ; "10"     ;
"0x281b06070400000101050 2a010300ea004020269aaa1060204010005ce" ;
    "0472826077" ;  "0x30343732473236303637" ; "10"     ;
"0x281b06070400000101050 2a010300ea004020269aaa1060204010006ce" ;

// end of file
```

Extract from the server scenario:

```
  <receive channel="channel-1">
    <action>
     ...
    </action>
    <primitive name="TC_BEGIN">
      <component name="TC_INVOKE" instance="InitialDP-data"
                  class="1"
                 operation-code="0"
                 operation-data="0x308188990000">
      </component>
    </primitive>
    <action>
```

The action portion goes on with the storage of dialogue portion elements:



And now, the server will restore the values in the header of the TC_CONTINUE primitive that it is sending right after:

```
    </action>
  </receive>

  <send channel="channel-1">
    <action>
     ...
    <!-- dlg portion BEGIN -->

      <restore name="DLG-APP-CTXT-NAME" entity="dlg-app-ctxt-name"></restore>
      <!-- dlg-app-ctxt-name is 0x060704000001003203 -->

      <restore name="DLG-USER-INFORMATION" entity="dlg-user-information"></restore>
      <!-- dlg-user-information is 28 1b 06 07 04000001010502 a010300e a0 04020269aa a1
060204010000ce -->

      <restore name="DLG-USER-INFORMATION-DEST"
      entity="dlg-user-information"
      begin="16" end="23"></restore>
      <!-- dlg-user-information is 28 1b 06 07 04000001010502 a010300e a0
060204010000ce 0204010000ce -->

      <restore name="DLG-USER-INFORMATION-A1"
      entity="dlg-user-information"
      begin="23" end="24"></restore>
```

```
      <!-- dlg-user-information is 28 1b 06 07 04000001010502 a010300e a0
060204010000ce a1 04010000ce -->

      <restore name="DLG-USER-INFORMATION-ORIG"
      entity="dlg-user-information"
      begin="24" end="29"></restore>
      <!-- dlg-user-information is 28 1b 06 07 04000001010502 a010300e a0
060204010000ce a1 04020269aa -->

    <!-- dlg portion END -->

    </action>
    <primitive name="TC_CONTINUE">
      <component name="TC_INVOKE"  instance="InitialDP-2-data"
                 class="2"
                 timeout="60000"
                 invoke-id="1"
                 operation-code="18"
                 operation-data="0x3081889900501400">
      </component>

      <component name="TC_INVOKE"  instance="InitialDP-3-data"
                 class="2"
                 timeout="60000"
                 invoke-id="2"
                 operation-code="23"
                 operation-data="0x308188990050163030">
      </component>
    </primitive>
  </send>
```

Note:
It is not possible to use only the user information, without setting the application context name. If you do so, you will get an error.

Note:
It is possible to use only the application context name.

5. Configuration files

There are 3 different configuration files:

- Generic configuration file - describing traffic and network parameters
- Protocol dictionary configuration file - rarely to be edited
- Scenario file - description of the message exchanges

5.1. Generic configuration

The generic configuration file describes the network environment as well as traffic parameters.

The network environment is described through "transport channel entities". The transport entity is then used as an attribute of `send` and `receive` scenario commands, as well as during the opening of the transport channel (see below).

```
<!-- TCAP example -->
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration name="Simple TCAP Client Conf">
```

```
<define entity="transport"
    name="trans-octcap"
    file="libtrans_octcap.so"
    create_function="create_ctransoctcap_instance"
    delete_function="delete_ctransoctcap_instance"
    init-args="flavour=WBB">
</define>
<!-- Then you specify the opening of the channel, on the transport previously
described. -->

<!-- For a server connected to SS7_Stack_1 on SSN 10, it will look like this: -->

<define entity="channel"
    name="channel-1"
    protocol="octcap-itu"
    transport="trans-octcap"
    open-args="class=SS7_Stack_1;ossn=10;application=1;instance=1">
</define>

<!-- For a client connected to SS7_Stack_2 on SSN 20, it will look like this: -->
<define entity="channel"
    name="channel-1"
    protocol="octcap-itu"
    transport="trans-octcap"
    open-args="class=SS7_Stack_2;ossn=20;application=2;instance=2">
</define>
```

5.2. TCAP dictionary

TCAP messages and parameters are described in XML dictionaries. The tool comes with a complete set of TCAP dictionaries: one for ITU and one for ANSI.

A dictionary contains several XML sections

5.2.1. Types

The "types" section doesn't contain any definition of type. The basic types already exist, such as number, string, etc.

5.2.2. Header

"header" section contains the description of message header. For TCAP, this is:

```
<header name="primitive" type="type"
        create-function="create_primitive"
        delete-function="delete_primitive">

  <fielddef name="type"         type="number"
            set-function="set_primitive_type"
            get-function="get_primitive_type">
  </fielddef>

  <fielddef name="d-address-pc"  type="number"
            set-function="set_primitive_d_address_pc"
            get-function="get_primitive_d_address_pc"
            config-field="dest-address-pc">
  </fielddef>

  <fielddef name="d-address-ssn" type="number"
            set-function="set_primitive_d_address_ssn"
            get-function="get_primitive_d_address_ssn"
```

```
          config-field="dest-address-ssn">
</fielddef>

<fielddef name="o-address-pc"  type="number"
          set-function="set_primitive_o_address_pc"
          get-function="get_primitive_o_address_pc"
          config-field="orig-address-pc">
</fielddef>

<fielddef name="o-address-ssn" type="number"
          set-function="set_primitive_o_address_ssn"
          get-function="get_primitive_o_address_ssn"
          config-field="orig-address-ssn">
</fielddef>

<fielddef name="uid"  type="number"
          set-function="set_primitive_uid"
          get-function="get_primitive_uid">
</fielddef>

<fielddef name="pid" type="number"
          set-function="set_primitive_pid"
          get-function="get_primitive_pid">
</fielddef>

<fielddef name="dlg-app-ctxt-name" type="string"
          set-function="set_primitive_appl_context_name"
          get-function="get_primitive_appl_context_name">
</fielddef>

<fielddef name="dlg-user-information" type="string"
          set-function="set_primitive_user_info"
          get-function="get_primitive_user_info">
</fielddef>

<fielddef name="dlg-abort-reason" type="number"
          to-string="primitive_dlg_abort_reason_to_string"
          from-string="primitive_dlg_abort_reason_from_string"
          set-function="set_primitive_abort_reason"
          get-function="get_primitive_abort_reason"
          default="NO">
</fielddef>

<fielddef name="p-abort-cause" type="number"
          to-string="primitive_p_abort_cause_to_string"
          from-string="primitive_p_abort_cause_from_string"
          set-function="set_primitive_p_abort_cause"
          get-function="get_primitive_p_abort_cause">
</fielddef>

<fielddef name="u-abort-cause" type="number"
          set-function="set_primitive_u_abort_cause"
          get-function="get_primitive_u_abort_cause">
</fielddef>

<!--  Report or  Notice -->
<fielddef name="report-cause" type="number"
          to-string="primitive_report_cause_to_string"
          from-string="primitive_report_cause_from_string"
          set-function="set_primitive_report_cause"
          get-function="get_primitive_report_cause">
</fielddef>

<fielddef name="termination" type="number"
          to-string="primitive_termination_to_string"
          from-string="primitive_termination_from_string"
```

```
            set-function="set_primitive_termination"
            get-function="get_primitive_termination"
            default="BASIC">
  </fielddef>

</header>
```

Note:
About the fielddef "termination": the default value is "BASIC". It can also be set to "PREARRANGED". This value can be changed in the scenario.

5.2.3. Body

"body" section contains the description of message body (which naturally comes after the header). For TCAP, this is an ANSI example:

```
<body name="component" type="component-type"
      create-function="create_component"
      delete-function="delete_component"
      value-field="operation-data"
      add-function="add_component"
      get-function="get_component">

     <fielddef name="component-type" type="number"
               set-function="set_component_type"
               get-function="get_component_type">
     </fielddef>

     <fielddef name="class" type="number"
               set-function="set_component_class"
               get-function="get_component_class"
               default="1">
     </fielddef>

     <fielddef name="timeout" type="number"
               set-function="set_component_timeout"
               get-function="get_component_timeout"
               default="30000">
     </fielddef>

    <fielddef name="operation-code-tag" type="number"
           to-string="component_operation_code_tag_to_string"
           from-string="component_operation_code_tag_from_string"
           set-function="set_component_operation_code_tag"
           get-function="get_component_operation_code_tag"
           default="NATIONAL">
    </fielddef>

     <fielddef name="operation-code" type="number"
               set-function="set_component_code"
               get-function="get_component_code">
     </fielddef>

     <fielddef name="operation-data" type="string"
               set-function="set_component_data"
               get-function="get_component_data">
     </fielddef>

     <fielddef name="invoke-id" type="number"
               set-function="set_component_invoke_id"
               get-function="get_component_invoke_id">
     </fielddef>

     <fielddef name="error-code" type="number"
               set-function="set_component_error_code"
```

```
                get-function="get_component_error_code">
    </fielddef>

    <fielddef name="problem-code" type="number"
              to-string="component_problem_code_to_string"
              from-string="component_problem_code_from_string"
              set-function="set_component_problem_code"
              get-function="get_component_problem_code">
    </fielddef>

   <fielddef name="problem-code-identifier" type="number"
             to-string="component_pb_code_identifier_to_string"
             from-string="component_pb_code_identifier_from_string"
             set-function="set_component_problem_code_identifier"
             get-function="get_component_problem_code_identifier">
   </fielddef>

    <fielddef name="correlation-id" type="signed"
              set-function="set_component_correlation_id"
              get-function="get_component_correlation_id">
    </fielddef>

    <fielddef name="parameter-type" type="number"
             to-string="component_parameter_type_to_string"
             from-string="component_parameter_type_from_string"
             set-function="set_component_parameter_type"
             get-function="get_component_parameter_type">
    </fielddef>

</body>
```

Warning:

Be careful with the fielddef "class". The default value is "1" and the possible values are:
* 1: class 1 both RESULT and ERROR may be sent back,
* 2: class 2 only ERROR may be sent back,
* 3: class 3 only RESULT may be sent back,
* 4: class 4 neither RESULT nor ERROR may be sent back.

This parameter in significant only in a send command.

Note:

About the fielddef "operation-code-tag": the default value is "NATIONAL" for ANSI and "LOCAL_TYPE" for ITU. You can also set it to "PRIVATE" for ANSI, and to "GLOBAL_TYPE" for ITU. This value can be changed in the scenario.

5.2.4. Dictionary

"dictionary" section contains all possible parameters that a message can contain. Here is a description for TCAP:

```
<dictionary>
  <component>
     <define name="TC_INVOKE">
        <setfield name="component-type" value="0"></setfield>
     </define>
     <define name="TC_RESULT_L">
        <setfield name="component-type" value="1"></setfield>
     </define>
     ...
  </component>

  <primitive session-id="uid" out-of-session-id="">
     <define name="TC_BEGIN">
        <setfield name="type" value="1"></setfield>
     </define>
```

```
    <define name="TC_CONTINUE">
       <setfield name="type" value="2"></setfield>
    </define>
    <define name="TC_END">
       <setfield name="type" value="3"></setfield>
    </define>
    ...
  </primitive>
</dictionary>
```

5.2.5. Configuration parameters

"configuration-parameters" section contains all possible TCAP parameters that can be defined in the configuration file. For each parameter a default value can be defined. You can also specified if the parameter is mandatory or not. Here is an example of TCAP configuration parameters for ITU:

```
<configuration-parameters>

   <paramdef name="orig-address-pc" default="0"> </paramdef>
   <paramdef name="dest-address-pc" default="0"> </paramdef>
   <paramdef name="orig-address-ssn" default="0"> </paramdef>
   <paramdef name="dest-address-ssn" default="0"> </paramdef>
   <paramdef name="orig-routing-type" mandatory="true" default="SSN"> </paramdef>
   <paramdef name="dest-routing-type" mandatory="true" default="SSN"> </paramdef>
   <paramdef name="orig-gt" default="123"> </paramdef>
   <paramdef name="dest-gt" default="123"> </paramdef>
   <paramdef name="orig-gt-indicator" default="tc_gt_type1"> </paramdef>
   <paramdef name="orig-gt-translation" default="tc_t_unused"> </paramdef>
   <paramdef name="orig-gt-numbering" default="tc_unknown_num"> </paramdef>
   <paramdef name="orig-gt-nature" default="tc_subscriber_nb"> </paramdef>
   <paramdef name="dest-gt-indicator" default="tc_gt_type1"> </paramdef>
   <paramdef name="dest-gt-translation" default="tc_t_unused"> </paramdef>
   <paramdef name="dest-gt-numbering" default="tc_unknown_num"> </paramdef>
   <paramdef name="dest-gt-nature" default="tc_subscriber_nb"> </paramdef>
</configuration-parameters>
```

Here is an example of TCAP configuration parameters for ANSI:

```
<configuration-parameters>

   <paramdef name="orig-address-pc" default="0"> </paramdef>
   <paramdef name="dest-address-pc" default="0"> </paramdef>
   <paramdef name="orig-address-ssn" default="0"> </paramdef>
   <paramdef name="dest-address-ssn" default="0"> </paramdef>
   <paramdef name="orig-routing-type" mandatory="true" default="SSN"> </paramdef>
   <paramdef name="dest-routing-type" mandatory="true" default="SSN"> </paramdef>
   <paramdef name="orig-gt" default="123"> </paramdef>
   <paramdef name="dest-gt" default="123"> </paramdef>
   <paramdef name="orig-gt-indicator" default="tc_gt_type1"> </paramdef>
   <paramdef name="orig-gt-translation" default="tc_t_unused"> </paramdef>
   <paramdef name="orig-gt-numbering" default="tc_unknown_num"> </paramdef>
   <paramdef name="orig-gt-nature" default="tc_subscriber_nb"> </paramdef>
   <paramdef name="dest-gt-indicator" default="tc_gt_type1"> </paramdef>
   <paramdef name="dest-gt-translation" default="tc_t_unused"> </paramdef>
   <paramdef name="dest-gt-numbering" default="tc_unknown_num"> </paramdef>
   <paramdef name="dest-gt-nature" default="tc_subscriber_nb"> </paramdef>
   <paramdef name="parameter-type-set" default="SEQUENCE_TYPE"> </paramdef>
</configuration-parameters>
```

Note:

The configuration of the point codes (OPC, DPC), SSN (OSSN, DSSN) and global title (GT) is done in the configuration files. The default values are set in the dictionary. You can modify their values in the scenario. In this case, the values in the scenario overwrite those in the configuration files.

5.3. Actions in scenario commands for TCAP

The <send> and <receive> scenario commands include an <action> and <primitive> sections.

The <action> section can be placed before or after the <primitive> section.

Actions placed before the primitive (called "**pre-actions**") are executed before the message is actually sent or received. Actions placed after the primitive (called "**post-actions**") are executed after the message is sent or received.

For example, actions that can be placed before a primitive are actions to set the values of some ids or to set some fields using external values, before sending a message. Example:

```
<send channel="channel-1">
    <action>
       <set-value name="uid" format="$(client-uid-counter)"></set-value>
       <restore-from-external field="1" entity="TC_INVOKE" sub-entity="operation-data"
                      begin="5" end="10"></restore-from-external>
    </action>
```

Actions that can be placed after a primitive are actions to store some values after the message has been received, or to check some values. Example:

```
<action>
       <store name="DLG-USER-INFORMATION-ORIG"
       entity="dlg-user-information"
       begin="16" end="21"></store>
       <check-value name="d-address-pc" behaviour="error">
       </check-value>
       <check-value name="o-address-pc" behaviour="error">
       </check-value>
    </action>
```

The list of possible actions is available in the reference section.

6. Reference

This section is the reference for all values and parameters of Seagull.

6.1. Generic configuration reference

6.1.1. TCAP configuration

This table is a list of traffic parameters applicable only to TCAP. Those parameters are present in the generic configuration file.

Name	Description	Recommended value	Example
dest-routing-type	It defines the way the messages are routed for outgoing messages. Possible values are: GT (routing on Global Title), SSN (routing on SSN only), GT_SSN (routing on Global Title and SSN) and DPC_SSN (routing on DPC and SSN).		<define entity="config-param" name="dest-routing-type" value="DPC_SSN"> </define>

orig-routing-type	It defines the way the messages are routed for incoming messages. Possible values are: GT (routing on Global Title), SSN (routing on SSN only), GT_SSN (routing on Global Title and SSN) and DPC_SSN (routing on DPC and SSN).		<define entity="config-param" name="orig-routing-type" value="DPC_SSN"> </define>
orig-address-pc	It defines the originating adress point code. You can modify this value in the scenario. In this case, the value in the scenario overwrites the one in the configuration files. Be careful in ANSI: this value has to be given as an integer and not in the format a.b.c. For example: 1.1.10 -> 0x01010a=65802.		<define entity="config-param" name="orig-address-pc" value="2"> </define>
dest-address-pc	It defines the destination adress point code. You can modify this value in the scenario. In this case, the value in the scenario overwrites the one in the configuration files. Be careful in ANSI: this value has to be given as an integer and not in the format a.b.c. For example: 2.2.57 -> 0x020239=131641.		<define entity="config-param" name="dest-address-pc" value="1"> </define>
orig-address-ssn	It defines the originating adress SSN. You can modify this value in the scenario. In this case, the value in the scenario overwrites the one in the configuration files.		<define entity="config-param" name="orig-address-ssn" value="20"> </define>
dest-address-ssn	It defines the destination adress SSN. You can modify this value in the scenario. In this case,		<define entity="config-param" name="dest-address-ssn" value="10"> </define>

	the value in the scenario overwrites the one in the configuration files.		
orig-gt	It defines the originating global title number.		<define entity="config-param" name="orig-gt" value="123"> </define>
dest-gt	It defines the destination global title number.		<define entity="config-param" name="dest-gt" value="123"> </define>
orig-gt-indicator	It indicates the type of the originating global title number.		<define entity="config-param" name="orig-gt-indicator" value="tc_gt_type1"> </define>
orig-gt-translation	It indicates the type of translation for the originating global title number.		<define entity="config-param" name="orig-gt-translation" value="tc_t_unused"> </define>
orig-gt-numbering	It indicates the type of numbering for the originating global title.		<define entity="config-param" name="orig-gt-numbering" value="tc_unknown_num"> </define>
orig-gt-nature	It indicates the nature of the originating global title.		<define entity="config-param" name="orig-gt-nature" value="tc_subscriber_nb"> </define>
dest-gt-indicator	It indicates the type of the destination global title number.		<define entity="config-param" name="dest-gt-indicator" value="tc_gt_type1"> </define>
dest-gt-translation	It indicates the type of translation for the destination global title number.		<define entity="config-param" name="dest-gt-translation" value="tc_t_unused"> </define>
dest-gt-numbering	It indicates the type of numbering for the destination global title.		<define entity="config-param" name="dest-gt-numbering" value="tc_unknown_num"> </define>
dest-gt-nature	It indicates the nature		<define

	of the destination global title.		entity="config-param" name="dest-gt-nature" value="tc_subscriber_nb"> </define>
parameter-type-set	Parameter set indentifier. ANSI only. This parameter's default value is "SEQUENCE_TYPE". It can also be set to "SET_TYPE".	SEQUENCE_TYPE	<define entity="config-param" name="parameter-type-set" value="SEQUENCE_TYPE"> </define>
discon-on-err	Disconnect on transport error.		<define entity="config-param" name="discon-on-err" value="no"></define>

Table 1: List of TCAP traffic parameters (traffic-param entity)