## Contents

Preface ............................................................................................................................................................................ v  
   Audience ........................................................................................................................................................................ v  
   Related Documents ....................................................................................................................................................... v  
   Conventions .................................................................................................................................................................... v  

1 Using the slpit Utility  
   About the slpit Utility ....................................................................................................................................................... 1-1  
   Running the slpit Utility ................................................................................................................................................... 1-2  
      Using the slpit Command ............................................................................................................................................ 1-3  
          Command Line Options ........................................................................................................................................ 1-3  
          Output Options ...................................................................................................................................................... 1-5  
      Running slpit in a Separate SLEE ............................................................................................................................ 1-6  
      Exit Codes ............................................................................................................................................................... 1-6  
   Understanding Script File Processing .............................................................................................................................. 1-7  
      Understanding TCAP primitives ................................................................................................................................ 1-8  
      Receiving Expected Operations .................................................................................................................................. 1-8  
      Managing Dialogs ....................................................................................................................................................... 1-9  
          Specifying a Particular Dialog .................................................................................................................................. 1-9  
          Ending a SLEE Dialog ........................................................................................................................................ 1-9  
          Aborting a SLEE Dialog ........................................................................................................................................ 1-9  
      Call Completion .................................................................................................................................................... 1-9  
      Cancelling slpit ....................................................................................................................................................... 1-10  
      Sending an Error to ACS ........................................................................................................................................... 1-11  
   The Script File ............................................................................................................................................................. 1-12  
      Call Messages .......................................................................................................................................................... 1-18  
      Example Scripts ....................................................................................................................................................... 1-50  

A About the SS7 Protocol Suite  
   The INAP Protocol ................................................................................................................................................... A-1  
   The CAP Protocol ...................................................................................................................................................... A-2  
   The MAP Protocol ...................................................................................................................................................... A-2  
   The IS-41 Protocol ...................................................................................................................................................... A-2  
   The TCAP Protocol ...................................................................................................................................................... A-2  
   The SCCP Protocol ...................................................................................................................................................... A-2  
   The M3UA Protocol ..................................................................................................................................................... A-3  


The SUA Protocol ................................................................................................................................. A-3
The SCTP Protocol ............................................................................................................................... A-3
The Internet Protocol ......................................................................................................................... A-3
Preface

This guide describes how to use the split utility for the Oracle Communications Network Charging and Control (NCC) software.

Audience

This document is intended for NCC network operators, system administrators and system integrators who have a need to do functional testing of applications, moderate load testing, and external interface testing.

Related Documents

For more information, see the following documents in the Oracle Communications Network Charging and Control 5.0.1 documentation set:

- Oracle Communications Network Charging and Control System Administrator’s Guide
- Oracle Communications Network Charging and Control User’s Configuration Guide
- Oracle Communications Network Charging and Control Service Logic Execution Environment Technical Guide

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><code>monospace</code></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
This chapter describes the Oracle Communications Network Charging and Control (NCC) `slpit` utility and how to run it and create the script file that defines call instances. The name `slpit` stands for Service Logic Program Instance Tester. The `slpit` utility sends and receives Intelligent Network Application Part (INAP) operations and acts as an interface to the Transaction Capabilities Application Part (TCAP) protocol.

Appendix A gives a brief overview of the Signalling System 7 (SS7) protocol suite, of which INAP and TCAP are a part.

About the slpit Utility

The `slpit` utility is a testing tool that you can use to do functional testing of NCC applications without concern for the protocol of a given network. From the perspective of the test application, the `slpit` utility is a real interface that converts the network messages to and from G8-INAP. It communicates with the application by way of the Service Logic and Execution Environment (SLEE), just like a regular interface.

Note: In this context, application or NCC application refers to the SLEE process to which the `slpit` utility is communicating. Usually, this is either `slee_acs`, which is the main ACS process, or `xmsTrigger`, which is the main XMS process. But it can also be `m3uaf`, which is also a SLEE process. The `m3uaf` process is further described in this document.

The `slpit` utility processes operations from an input text file rather than a real network. The input text file is called the script file and it is a file that you create. In the script file, you add commands and send and receive operations that define the call sequences that you want to test. See “The Script File” for more information.

The utility parses the responses from the test application and compares them to the responses that the script file expects.

The `slpit` utility has the following characteristics:

- It allows you to effectively test Intelligent Network (IN) applications without requiring a physical telephony network or a low-level network-specific test tool.
- You can use it to do functional testing of NCC applications without concern for a particular network protocol. As long as the application provides the correct functionality in G8-INAP, you can assume that it will perform the same way on a particular network with the appropriate interface.
Running the split Utility

It acts as a normal TCAP interface to trigger IN platform service logic, providing emulation of a service switching point (SSP) and specialized resource function (SRF) interactions.

It supports the following IN protocols: CAP, MAP, SCCP, GPRS and IS41.

It uses a script file in which you define the INAP operations that are sent and received for one or more types of calls. A single instance of split can run many call instances and many calls can be in-progress at once. A call is initiated by the script and different distributions and rates are possible. Multiple protocols are supported.

You can use it to do moderate load testing and external interface testing, in addition to using it for functional testing of applications.

You can run it in the same SLEE as the application being tested or in a separate SLEE using appropriate TCAP interfaces.

On a production NCC system, the slee_acs process and the xmsTrigger process communicate with a process called m3ualf, using a TCAP-like protocol. The m3ualf process is also a SLEE process. The m3ualf process turns the TCAP-like events into messages that are sent over the IP network in a protocol stack that consists of one of the following:

- MAP over TCAP over SCCP over M3UA over SCTP over IP
- CAP over TCAP over SCCP over M3UA over SCTP over IP

There are two ways to run the split utility. In the first way it replaces the m3ualf process so that it communicates with slee_acs and xms_Trigger but does not send anything over the IP network. This allows you test the higher layers of a protocol but does not include any processing that would normally happen inside m3ualf.

The second way to run the split utility allows you to test certain functions that happen in the m3ualf process. To run the utility this way, you must configure two machines. For example, if you configure machines SLC1 and SLC2, you configure SLC1 exactly like a production SLC, with slee_acs and xmsTrigger talking to m3ualf. You configure SLC2 with only m3ualf and the split utility on it. See "Running split in a Separate SLEE" for information about running the split utility in this configuration.

Running the split Utility

The split utility processes a script file in which you define the progress of call instances through a series of INAP send and receive operations.

The split utility creates a distribution with a list of call types and other parameters that control the launching of calls, which is known as the call rate, and the terminating condition of the distribution, which is generally the number of calls launched. A distribution processes the call types in a round-robin fashion until the completion condition is met. The split utility allows you to create the following types of distributions:

- A uniform distribution has an interval and a total call count. The interval specifies the number of seconds that are to elapse before launching each call until the total number of calls is reached. The minimum interval is around a tenth of a second. For example, the following startcall line would run the call every .5 seconds for a total of 10 times:

  startcall using uniform 0.5 10
A **poisson** distribution has a lambda value and a total call count. The lambda variable represents the average interval between calls rather than the exact interval.

The **once** distribution launches one of each specified call type immediately.

A **once** distribution will run through the contents of the test script once and exit with a result of **SUCCESS**, **FAILED** or **ABORT**.

The type of distribution is determined by the type of testing that you are performing. You specify the distribution type in the script file using the `startcall` command, for example:

```
startcall <id> using <distribution>
```

So for a script in which you wanted to run only one call that was started with `define call 982 {}`, you would have start the call with a line like the following:

```
startcall 982 using once
```

---

### Using the `slpit` Command

The `slpit` utility is located in the following directory:

```
/IN/service_packages/TEST_TOOLS/bin
```

The basic command for running the `slpit` utility specifies a service and the name of the script input file. Additional command line options allow you to request validation of the script file, define a global variable, specify the debug output, specify output options, and perform various other actions.

#### Command Line Options

You can run the `slpit` utility with the following commands:

```
slpit [options...] [<script>]
```

- `-V`
- `-h`
- `-a`

The only command line option that is typically required to run `slpit` is `-k`, which allows you to specify a service key other than the arbitrary default of 101. The following example shows the simplest command to run `slpit` with a script file:

```
slpit -k 1 my_script_file
```

You can also provide the script on `stdin` as shown here, although in this case `slpit` will not know the name of the script:

```
slpit -k 1 < my_script
```

When you are trying to correct syntax errors in a script, the `-c` option is useful because it causes `slpit` to exit immediately after parsing, without running any calls:

```
slpit -c my_script_file
```

The following command line options allow you to request special processing by `slpit`.

- `-a`

Act as an application instead of the default interface. For more information, see "Running `slpit` in a Separate SLEE".
Running the slpit Utility

- **-c**
  Validate the script file and exit.

- **-C csvfile**
  Writes the following values to the specified comma-separated values (CSV) file every ten seconds: the time, calls per second (CAPS), and outstanding call count.

- **-D name=value**
  Predefines a global variable with the specified name and value, where *name* is the name of the variable and *value* may be an integer or a double quoted string. Defining a global variable could be useful for making a change to the script easier in the future. For example, you could define the destination phone number and then refer to it in the call definition using the variable. Then, in the future, when you want to define a new call with a different number, you would only need to change the number in one place.

- **-d level**
  Sets the level of debug output. Valid range of levels is 0 to 5, with 0 indicating no output and 5 indicating the maximum level.

- **-g**
  Makes the utility more tolerant of errors, causing it to continue, if possible, rather than abort.

- **-h**
  Prints version and build information, like the `-V` option, plus a summary of the usage information.

- **-k key**
  Initiate calls with a service key value of *key* rather than 101, which is the default. These are the service keys defined in the `SLEE.cfg` file.

- **-i interval**
  Report call summary information at the interval specified by *interval*, which is a number of seconds.

- **-I name**
  Adds the value of *name* as a suffix to the interface name. This option is required if you run more than one instance of slpit simultaneously in the same SLEE.

- **-M interval**
  Can be used with the `-m` option to write average timing information per primitive to the CSV file at the interval specified by *interval*. If the specified interval is 0, the average timing information will be written when the script completes. This parameter works only if the slpit script expects a response because the average durations cannot be calculated otherwise.

- **-m directory**
  Enables logging of timestamps per TCAP primitive for messages sent and received. The utility writes the information by call type to a comma-separated values (CSV) file in the specified directory.

- **-O flags**
  Enables the specified output flags. See "Output Options" for more information.
-o level
Sets the level of normal output. Valid range of levels is 0 to 5, with 0 indicating no output and 5 indicating the maximum level. The default is 3, which produces a reasonable amount of output that is not excessive.

-p protocol
Sets the preferred TCAP protocol in case there is a conflict between the INAP/CAP and MAP tag values, as there is some overlap. Valid values are map, is41, and inap, which is the default.

-R
Recreates the main dialog, if it no longer exists, using the last received originating reference as a correlation ID. This is required for the CAP3 GPRS message sequence. This option does not work if the -a option is also submitted.

-T
Enables the use of the SLEE Timer interface for delays between sending new requests, or responses to inbound requests. Without this option, delays are handled by polling. Use of this option is not recommended. The Timer interface is not ideal for this purpose.

-v
Enables verbose output, setting the output level to the maximum. This is equivalent to setting the -o option to 5.

-vv or -v -v
Sets the debug output level to the maximum.

-V
Prints version and build information and then exits.

Output Options
In addition to the overall output level that is controlled by the -o option, you can enable the following more specific output features with the -O option. Most of these are enabled automatically at various numeric output levels.

calldefntrace
Displays a brief summary of the call definition at each step in the call execution, including an indication of the current step. Automatically enabled at overall output level 4.

callrate
Displays the average call rate achieved before the slpit utility terminates. It calculates the call rate by dividing the total run time by the number of calls started.

callsummary
Prints a table summarizing the number of calls run, the number of successful calls and the number of partially and totally failed calls. Call types that had at least one aborted call are marked with four asterisks (****); call types with failed calls are marked with a single asterisk (*).

fullcallsummary
Prints a more detailed call summary table than the one produced by the callsummary option. The information is the same as produced by the callsummary option; the format of the table produced by callsummary is just more concise.
**triptiming**
Records and displays round-trip message times for each call.

**sleecheck**
Checks changes in the SLEE free resource object counts at the end of the run, and if there are any changes, displays a table of the free counts. The resources are SLEE resources such as calls, dialogs, events, and application instances. The rows for resource types that showed a positive delta are marked with a single (*); those with a negative delta, which indicates a potential memory leak, are marked with four asterisks (****). A positive delta in free resource counts indicates that running one or more calls caused resources to be freed. This is not uncommon with Advanced Control Services (ACS), which is prompted to free a SLEE management event when the first call event arrives.

**parsedebug**
Enables the extremely verbose debug output for the GNU Bison parser. This option is not automatically enabled at any output level because it is useful primarily when debugging the parser.

### Running slpit in a Separate SLEE

When running the slpit utility in a separate SLEE, you must specify the `-a` and `-k` command line options. The `-k` option must specify the SLEE service key that is assigned to the m3uaIf process in the SLEE.cfg file on the machine where the m3uaIf process is running.

### Exit Codes

Table 1–1 describes the exit codes that the slpit utility uses to indicate whether or not the run was successful. The slpit utility writes the exit code to stdout (standard out), which you can redirect to a file, if you wish.

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Execution completed successfully</td>
</tr>
<tr>
<td>1</td>
<td>General or usage error, which usually indicates that the command line options were not valid.</td>
</tr>
<tr>
<td>2</td>
<td>Script parsing error. Either slpit could not read the script file or it encountered a syntax error in the script. The utility displays diagnostics on stderr.</td>
</tr>
<tr>
<td>4</td>
<td>The initial connection to the SLEE failed, most likely because no SLEE is running.</td>
</tr>
<tr>
<td>5</td>
<td>A SLEE entity that slpit required could not be contacted. This can occur when slpit is directed to use the Timer interface for running timers but the interface could not be found.</td>
</tr>
<tr>
<td>6</td>
<td>Call creation failed. From most likely to least likely, the possible reasons include: the service key for the call being created is not configured; a resource for a SLEE dialog or for call instances has been exhausted; the SLEE for the service configured on the service key never started or has been stopped.</td>
</tr>
<tr>
<td>10</td>
<td>At least one call instance failed. There were no errors that prevented slpit from running to completion, but at least one call instance ended in the FAILED state.</td>
</tr>
</tbody>
</table>
Understanding Script File Processing

The script file is an input file in which you define the call instances that you want the \texttt{slpit} utility to process. Call instances are defined with a set of commands and INAP operations that you specify in sets of \texttt{send} and \texttt{receive} messages. The following example shows the beginning of a call instance definition:

\begin{verbatim}
define call assisting_ip_pa {
  SERVICE_NUMBER ?= "555801"
  send {
    initialdp
    calledpartynumber SERVICE_NUMBER
    callingpartynumber "40002000"
    callingpartyscategory 10
    locationnumber "40002000"
    eventtypebcsm analyzedinformation
  }
  receive {
    establishtemporaryconnection
    address "1234"
  }
}
\end{verbatim}

You start call processing by including a \texttt{startcall} command, like the one shown in this example:

\begin{verbatim}
startcall assisting_ip_pa using once
\end{verbatim}

The \texttt{slpit} utility can reference the call types that you define in the script file only after the script file has been parsed. Starting a call creates a distribution but the distribution does not start creating new call instances until script processing completes.

When you run the \texttt{slpit} utility, it processes all distributions and calls in the script file before stopping.

In general, each call that \texttt{slpit} executes produces one call instance and one or two dialogs in the SLEE. The first dialog is called the \texttt{main} dialog. The second dialog, which will exist only for parts of some calls, is referred to as the \texttt{assisting} dialog.

The first message sent for a call must be an InitialDP, or an appropriate TC\_BEGIN message. The first action in a call cannot be a receive message.

\begin{table}[h]
\centering
\caption{slpit Exit Codes (Cont.)}
\begin{tabular}{|c|p{8cm}|}
\hline
Exit Code & Description \\
\hline
11 & At least one call instance aborted. There were no errors that prevented \texttt{slpit} from running to completion, but at least one call instance ended in the ABORTED state. \\
\hline
\end{tabular}
\end{table}

\textbf{Note:} Sending an \texttt{AssistRequestInstructions} message creates a second dialog on the same call instance to simulate the dialog between the intelligent peripheral and the service control point (SCP).

This means that for the scenario in which \texttt{slpit} runs in the same SLEE as the application, the correlation ID for each dialog is not required to match. Normally, the TCAP interface would resolve the correlation ID to create the second dialog on the correct call instance.
The `slpit` utility does very little validation to ensure valid call flow. It primarily ensures that the dialog is handled correctly. For example, you do not get a warning if you forget to send an `ApplyChargingReport` message at the end of a monitored call, but you do get a warning, if you do not explicitly terminate a dialog.

### Understanding TCAP primitives

All TCAP messages are primitives although some primitives are not messages. That is, some primitives are transferred only inside the local machine. A TCAP primitive consists of one or more TCAP components and can be one of the types described in Table 1–2:

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unidirectional</td>
<td>A primitive with no primitives coming after; also known as a notice.</td>
</tr>
<tr>
<td>Begin</td>
<td>Begins a dialog with other primitives coming after it.</td>
</tr>
<tr>
<td>Continue</td>
<td>A subsequent primitive sent on an existing dialog with other primitives coming after it.</td>
</tr>
<tr>
<td>End</td>
<td>The last primitive; closes it’s dialog.</td>
</tr>
<tr>
<td>Abort</td>
<td>Closes the dialog due to an error.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Closes the dialog when the invoke timer expires without receiving a response. This is an example of a primitive that is not a message.</td>
</tr>
</tbody>
</table>

### Receiving Expected Operations

Each received message corresponds to a single TCAP primitive and can contain one or more INAP operations.

The received message must contain the expected INAP operations in the order specified in the `receive` message section of the call definition. See "Call Definition Commands and Messages" for more information about defining calls.

---

**Note:** Operations can come in one primitive but also can sometimes come in separate primitives, depending on the application or the service.
the most desirable behavior, however, because it does not cause a TCAP ABORT message to be sent the application, if it is running in the same SLEE. Therefore, if you expect a timeout, you should override the default with a more appropriate action.

Managing Dialogs

The send message includes options that allow you to specify a particular dialog on which to send and also to end a SLEE dialog. You can also use the abort primitive to abort a dialog and use other messages to send a error to ACS.

Specifying a Particular Dialog

To send operations specifically on either the main or assisting dialog, specify the dialog in the send primitive. For example, the following send primitive sends the operations on the assisting dialog.

```
send assisting {
  <operations>
}
```

To send on the main dialog, specify main of assisting.

Ending a SLEE Dialog

The easiest way to end a SLEE dialog is to include the end option on the last send message in the dialog, as shown in the following example:

```
send end {
  ...
}
```

When a call completes, whether it is successful or aborted, the slpit utility automatically closes any open dialogs. If the slpit utility runs in the same SLEE as the application, the application receives only an indication that the dialog is closed and might not handle it in the same way that it does the shutdown of a real TCAP dialog. Therefore, if a call ends with a status of Failed or Okay, and it has dialogs open, the slpit utility displays a warning message. If a call was aborted, you can assume it might have an open dialog.

The slpit utility terminates a dialog when the application sends or receives a terminating event. When the slpit utility ends a dialog, it writes a line of output that indicates the number of messages that are still in the queue. Usually, you can ignore these messages because only internal SLEE messages will be left.

Aborting a SLEE Dialog

You can also explicitly abort a dialog by using the abort message. Specify the open option to abort any open dialogs.

```
abort [main | assisting | open]
```

Call Completion

Each call instance finishes in one of the following states:

- Aborted

Execution of the call was interrupted because something was sufficiently wrong that the call could not or should not continue. For example: 1) an attempt was
made to send an event when a dialog was no longer available, or 2) a run-time error occurred.

- **Failed**
  
  The call was not completely successful but the errors were not sufficient to interrupt the call. The most likely cause is a discrepancy between the received and expected parameters for an operation. The call is failed but allowed to continue because the difference may not be significant.

- **Okay**
  
  The call completed without errors.

A call can finish for the following reasons:

- When execution reaches the end of the call definition; the final call state will be either Failed, or Okay depending on whether there were errors in the run.

- If a finish call command is executed, the call run is immediately finished either with its current state or the override state that is specified in the finish call command.

- If the slpit utility encounters a serious error, it aborts the call immediately.

### Cancelling slpit

You can run multiple calls with slpit, either by specifically defining each call in the script input or by using the uniform or poisson distribution models.

By default, the slpit utility does not stop generating calls if any call aborts or fails. You can change this behavior by using the cancel after command. You can place this command anywhere outside a call definition in a slpit script. The command has the following forms, each of which is self explanatory:

```plaintext
cancel after none
cancel after abort
cancel after failure
cancel after abort or failure
```

See "The Call Sequence" for more information about these commands.

You can also allow the slpit utility to continue after an abort or failure until a specified limit is reached.

You can use the following form of the cancel command to cancel a run after a specified number of failures or aborts occurs.

```plaintext
cancel after <number> [abort|failure] [or aborts|failures]
```

If the number of specified aborts or failures occurs for the call, this command causes the slpit utility to stop call processing and exit. The program accepts either abort or aborts. It also accepts either failure or failures.

The following command specifies a time limit in seconds to the number of failed or aborted calls that the slpit utility receives before it cancels call processing and exits:

```plaintext
cancel after <number> [aborts|failures] [or aborts|failures] in <number> seconds
```

The program accepts either second or seconds.
Sending an Error to ACS

You can send an error to ACS by using either the `tcapReject` message or the `error` message. See "tcapreject" and "error" for more information.
The content of an slpit script file consists of a few commands and a set of operations that define a call and the progress of a call.

Syntax

The following syntax conventions apply to the commands and operations that appear in the script file. Any other element is a literal part of the syntax.

[ ]
Square brackets indicate that the enclosed items are optional. For example, the correlationid parameter in the following operation is optional.

establishtemporaryconnection
  address <digits>
  [correlationid <digits>]

| A pipe separates one or more choices. For example, in the following finish call operation, you can optionally specify a final state of aborted, failed, or okay.

finish call [aborted|failed|okay]

...

An ellipsis indicates that an item can be repeated one or more times. In the following example, part must occur at least once but the ellipsis indicates that it can be repeated one or more times.

[variableparts <part> [<part>...]]

<>
Angle brackets enclose a named reference to another syntax element, most commonly an atomic token or basic data type such as <integer>, <string>, or <bcd>.

The slpit utility supports the following three styles of comments, which can appear anywhere in the script file:

//
C++ style comments that can extend to the end of a line. The following line illustrates a full line comment:

//This is a full line comment
calledpartynumber '049393520' // This is an in-line comment

#
Shell-style comments that can extend to the end of a line.

/*... */
C-style comments that can extend multiple lines between the beginning and ending delimiters.

Commands

You can include the following commands in the script file in addition to the messages and operations that define a call:
include <file>
Includes the named file in the slpit script, enabling you to include a call sequence that is defined in a separate file. The value of file includes the directory path to the file’s location.

define call <ID> { <call sequence> }
Defines a set of call sequence messages and operations. The ID is an identifier you assign to the call and use to reference the call in other commands. See “The Call Sequence” for more information on <call_sequence>.

startcall <id> [ <id...> ] using <distribution> [<seed>] maxconcurrent <limit> \
[after <wait_seconds>]
Defines the call types, the number of calls, and the call rate at which to start generating calls. You can start multiple call types and call rates by including multiple startcall commands in the script file.

For all distribution types, you can specify a random seed, which is a number that will be used to initialize the call rate. If not specified, the current clock time is used.

For all distribution types, you can also specify a maximum concurrent number of calls to hold open. This overrides any calls-per-second (CAPS) rate and causes a lower CAPS rate to be used. This is useful for specifying the maximum load that can be supported for the test system.

For all distribution types, you can specify that the block of calls are to be run after a wait time of a specified number of seconds. This is useful for specifying a stepped call rate, in which you define one startcall for each step, with each one timed to begin after the preceding one has finished.

After the keyword using, you can use the following forms of the command:

- uniform <delay> <count>
- once
- poisson <delay> <count>
- poisson <delay> <ramp> <count>

The <delay>, <count>, and <ramp> values must be defined as a number with a decimal point.

The <delay> parameter is the average interval between calls. You can also express this value as calls per second and you can do so by using the CPS keyword. For example 10.0 cps is equivalent to 0.1.

The first form of the poisson command generates calls at random with the average interval between calls specified by the <delay> parameter.

The second form of the poisson command ramps up from zero calls per second to 1/<delay> calls per second, taking about <ramp> seconds to reach the maximum call rate. It then flattens off at that rate.

For information on using the cancel command to cancel calls, see "Cancelling slpit".

---

**Call Definition Commands and Messages**

The basic format of a call definition looks like this:

```plaintext
define call <id> { 
  <call sequence>
}
```

Each call type is identified by an id that can be either a number or a name that starts with a letter and contains only letters, digits, and underscores.
The call sequence consists of a set of call definition messages that describe the progress of a call.

**The Call Sequence**

The call sequence consists of one or more of the following call definition messages:

```plaintext
send [end] [assisting|main] { <message details> ... }  
receive [assisting|main] { <response details> ... }  
allow receive abort assisting  
[send] abort [assisting|main|open]  
wait <delay>  
=id = <expression>  
=id ?= <expression>  
finish call [aborted|failed|okay]  
default timeout none  
default timeout <expression> [ { <new call sequence> } ]  
close [assisting|main|open]  
cancel after [none|abort|fail] [or [abort|fail] ]
```

The `slpit` `send` message sends an event containing one or more operations, as determined by the message details, which you can modify through the use of various flags.

```plaintext
send [end] [assisting|main] { <message details> ... }  
```

The `end` flag causes the messages to be sent as the final event on the dialog. You can use the `assisting` or `main` flag to override the dialog on which the message is sent. See "Understanding Script File Processing" for more information.

The `slpit` utility expects to receive an event containing one or more operations as determined by the response details in a `receive` message. You can use either the `assisting` or the `main` flag to override the dialog on which the message is expected to arrive.

```plaintext
receive [assisting|main] { <response details> ... }  
```

See "Understanding Script File Processing" for more information.

An `allow abort` message indicates that the `slpit` utility should expect an abort to arrive from TCAP on the specified dialog at some time in the future. This is different from `receive` in that the `slpit` utility doesn’t stop and wait for the abort, it just continues processing.

```plaintext
allow receive abort assisting  
```

An `abort` message causes the `slpit` utility to send a TCAP abort on the specified dialog or dialogs (the default is the main dialog). Specifying open dialogs causes the `slpit` utility to abort any dialogs still open for the call.

```plaintext
[send] abort [assisting|main|open]  
```

The `slpit` utility will pause its processing of the call for a specified delay or until it is interrupted by a received event. You can specify the delay as an integer value representing microseconds or as a floating point value representing seconds. The following example illustrates the format of the message:

```plaintext
wait <delay>  
```

The `slpit` utility can evaluate an expression and assign its value to a named variable. You can use the `?=` operator if there is not already a value bound to the given name, which can be useful for defining default values in parameterized scripts.
The finish call message finishes the call. You can specify a final state of 
\[\text{aborted|failed|okay}\] to override the established state. For example, finish call 
okay causes a failed call to be recorded as successful.

```bash
finish call [aborted|failed|okay]
```

The default timeout message specifies the default timeout that the slpit utility uses 
when waiting for a message. If you specify the \text{<new call sequence>} section, which is 
enclosed in curly braces, the call sequence will run when the timeout occurs rather 
than the lines that follow in the main call definition. If you specify none, it turns off the 
timeout altogether.

```bash
default timeout <expression> [ {<new call sequence>} ]
default timeout none
```

The close message closes the SLEE dialog by way of a DIALOG CLOSED event on the 
given dialog. If you use the open option, the slpit utility closes all open dialogs.

```bash
close [assisting|main|open]
```

The cancel message allows you to force the slpit utility to exit any call immediately 
upon the failure or abort of a call, if the call fails or aborts at some future point. This 
feature is most useful when running multiple calls in one slpit run, as when using the 
uniform and poisson call distribution models.

This message has the following four formats:

- **cancel after none**
  This version prevents the slpit utility from exiting the run on the abort or failure 
of the call.

- **cancel after abort**
  This version causes the slpit utility to stop call processing or generation and exit if the call aborts. You can substitute the word aborts for abort.

- **cancel after fail**
  This version causes the slpit utility to stop call processing and exit if the call fails. You can substitute the words failure, failures, and fails for fail.

- **cancel after abort or fail**
  This version causes the slpit utility to stop call processing or generation and exit if the call aborts or fails. You can substitute words as described in the other formats of the cancel message.

You can use multiple messages like this in the same call definition to provide for 
situations in which the call might not fail before a certain command, but could fail 
after another command.

You can also define a global cancellation strategy outside of a call definition. 
See “Cancelling slpit” for more information.

### About Expressions and Comparators

An expression generates a value that you can use, for example, as the parameter value 
for a send operation. The simplest form of expression is a constant value. For example,

```bash
<id> = <expression>
<id> != <expression>
```

You can use multiple messages like this in the same call definition to provide for 
situations in which the call might not fail before a certain command, but could fail 
after another command.

You can also define a global cancellation strategy outside of a call definition. 
See “Cancelling slpit” for more information.
"5551234" appearing in a slpit script is usually an expression generating a digit string. Some more complex expressions are supported:

- Expressions can use the value of a variable by name.
- Limited integer arithmetic is supported: subtraction, addition, and multiplication. Integer arithmetic expressions may also contain parentheses for grouping.

In the syntax descriptions that follow, <integer expression> indicates that an expression should appear at that point and the expression should produce an integer. The same applies for other comparator types like <number comparator>. For example:

```plaintext
callConnectedElapsedTime(talktime - 20) * 10
```

You can also specify ranges of numbers as an expression, including the Nature of Address of the generated numbers, which defaults to 3, if not specified:

```plaintext
CLI = RANGE(integer: NoA) "49900010001" "49900020001" SEQUENTIAL
```

or

```plaintext
CLI = RANGE(<integer: NoA>) "49900010001" "49900020001" RANDOM
```

You can also obtain values from a flat file, such as one for vouchers. Use the following syntax to specify a flat file:

```plaintext
VOUCHERNUM=FROM_FILE 'vouchers.txt'
```

This will take a line from vouchers.txt and use that value wherever VOUCHERNUM is used. If you want to randomly use rows from vouchers.txt, you need to randomize the file before you pass it to the slpit utility. Not having enough rows in your file to match the number of calls causes the slpit utility to produce an error and stop once the numbers have run out.

A comparator is a pattern for checking received values such as the parameters in received operations. There are three relatively simple comparators:

- any
- [=] <expression>
- <comparator> -> id

The any comparator matches any value.

The simplest comparator is simply an expression and tests for equality. You can optionally precede the expression with = to make the test explicit. Because the simplest expression is a constant value, comparators usually test for equality with a simple constant value. It might also be useful to compare to the value of a variable.

The last comparator is a special case. It generates a match or a mismatch based on the result of the comparator, which can be any other comparator, but also stores the value being checked in the variable named by id. This allows you to store a received parameter value for later use.

In the syntax description, <integer comparator> indicates that you can include any comparator at that point, but the comparison should be for an integer, so the expression or expressions underlying the comparison should generate integers. The same thing applies for other comparator types like <number comparator>.

```plaintext
<number>: [{{noa}}] <digits> [screening <integer>] [presind <integer>]
```
For outgoing numbers, the following default values are substituted for any field not specified:

\[\text{noa}=3 \quad \text{screening}=0 \quad \text{presind}=1 \quad \text{numberplan}=1 \quad \text{innorni}=0\]

For incoming numbers, any value is allowed for fields that have not been specified except \(<\text{digits}>\).
Call Messages

This document does not explain the semantics of INAP, MAP, or CAP operations, except where the mapping from the parameters in the script to those in the actual operations is not obvious. Please refer to the relevant standards documentation for the descriptions and procedures for particular operations. See the following standards documents for more information:

- Intelligent Network (IN); Intelligent Network Capability Set 1 (CS1); Core Intelligent Network Application Protocol (INAP); Part 1: Protocol specification. European Telecommunication Standard, ETS 300-374-1, September 1994
- 3rd Generation Partnership Project; Technical Specification Group Core Network; Customized Applications for Mobile network Enhanced Logic (CAMEL) Phase 4; CAMEL Application Part (CAP) specification (Release 5). 3GPP 29.978 5.4.0 (2003-06)
- Digital cellular telecommunications system (Phase 2+); Mobile Application Part (MAP) specification (GSM 09.02 version 7.5.0 Release 1998). ETSI TS 100 974 V7.5.0 (2000-07).

Call messages are divided into send message operations and receive message operations.

Send Message Operations

You can use the following operations in the send message portion of a call definition.

**alertServiceCentre [parameters]**

You can use this operation for MAP handling. It performs alerting between MSC and HLR and it has the following parameters in any order:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msisdn</td>
<td>&lt;number expression&gt;</td>
</tr>
<tr>
<td>serviceCentreAddress</td>
<td>&lt;number expression&gt;</td>
</tr>
</tbody>
</table>

**alertServiceCenterWithoutResult [parameters]**

You can use this operation for MAP handling. It performs alerting between MSC and HLR and it has the following parameter in any order:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msisdn</td>
<td>&lt;number expression&gt;</td>
</tr>
<tr>
<td>serviceCentreAddress</td>
<td>&lt;number expression&gt;</td>
</tr>
</tbody>
</table>

**anyTimeInterrogation [parameters]**

This operation performs information enquiry between GSM SCF and HLR. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Min.</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestedInfo</td>
<td>(locationInformation</td>
<td>subscriberState)</td>
<td>N/A</td>
</tr>
<tr>
<td>imsi</td>
<td>&lt;bcd&gt;</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>msisdn</td>
<td>&lt;bcd&gt;</td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>
The Script File

Using the slpit Utility

1-19

The following table lists the valid combinations of the fields that make up a global title:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Min.</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>qmScf</td>
<td>&lt;bcd&gt;</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>sccp_orig_pc</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>65535</td>
</tr>
<tr>
<td>sccp_orig_ssn</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>255</td>
</tr>
<tr>
<td>sccp_orig_tt</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>255</td>
</tr>
<tr>
<td>sccp_orig_np</td>
<td>&lt;integer&gt;</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>sccp_orig_noa</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>sccp_orig_rti</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>sccp_orig_digits</td>
<td>&lt;digits&gt;</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>sccp_dest_pc</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>65535</td>
</tr>
<tr>
<td>sccp_dest_ssn</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>255</td>
</tr>
<tr>
<td>sccp_dest_tt</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>255</td>
</tr>
<tr>
<td>sccp_dest_np</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>sccp_dest_noa</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>sccp_dest_rti</td>
<td>&lt;integer&gt;</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>sccp_dest_digits</td>
<td>&lt;digits&gt;</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Global Title Type Fields

<table>
<thead>
<tr>
<th>Global Title Type</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>noa, digits</td>
</tr>
<tr>
<td>2</td>
<td>tt, digits</td>
</tr>
<tr>
<td>3</td>
<td>tt, np, digits</td>
</tr>
<tr>
<td>4</td>
<td>tt, np, noa, digits</td>
</tr>
</tbody>
</table>

Note that for global title types 3 and 4, the encoding is always binary coded decimal (BCD), that is 1 if there is an odd number of digits and 2 if there is an even number of digits.

**applychargingreport**

This operation provides feedback from the service switching function (SSF) to the service control function (SCF). It has the following format:

```
applychargingreport
    thresholdtime <integer>
    endofcallindicator <integer>
    [freecallindicator <integer>
```

The following format is available for use with INAP CAMEL extensions:

```
applychargingreport
    receivingSide <number>
    [timeNoTariffSwitch <number> |
    timeSinceTariffSwitch <number> |
    timeSinceTariffSwitch <number> tariffSwitchInterval <number> ]
    [ callActive <number> ]
```
applyChargingReportGprs
This operation provides a report from the GPRS SCF to the GSM SSF. It has the following format:
applyChargingReportGprs
  ( gprsvolumeifnottariffswitch <integer> | gprsvolumesincelasttariffswitch <integer> [ gprstariffswitchinterval <integer> ] | gprstimeifnottariffswitch <integer> | gprstimesincelasttariffswitch <integer> [ gprstariffswitchinterval <integer> ] )
  <qos-list>
    gprsActive <integer>
    [ gprsPdPd <integer> ]
    [ <ChargingRollover> ]

The <qos-list> section is one or more of the following in any order:
  gprsrequestedqos <gprs-info>
  gprsnegotiatedqos <gprs-info>
  gprssubscribedqos <gprs-info>

The <gprs-info> data is the same as defined in intialDpGprs. See "InitialDpGprs" for more information.

The <ChargingRollover> section is optional, consisting of either:
  <TransferredVolumeRollOver> | <ElapsedTimeRollOver>

The <TransferredVolumeRollOver> parameter consists of a choice of:
  gprsvolumeifnottariffswitch <integer> | gprsvolumesincelasttariffswitch <integer> (optional)
  gprsvolumetariffswitchinterval <integer> (optional)

Where <ro-VolumeIfTariffSwitch> consists of a sequence of:
  gprsvolumesincelasttariffswitch <integer> (optional)
  gprsvolumetariffswitchinterval <integer> (optional)

The <ElapsedTimeRollOver> consists of a choice of:
  gprstimeifnottariffswitch <integer> | gprstimesincelasttariffswitch <integer> (optional)
  gprstimetariffswitchinterval <integer> (optional)

Where <ro-TimeIfTariffSwitch> consists of a sequence of:
  gprstimesincelasttariffswitch <integer> (optional)
  gprstimetariffswitchinterval <integer> (optional)

applyChargingReportAckGprs
This operation has no parameters.
applyChargingReportAckGprs

assistrequestinstructions
This operation is used by the SSF to report a specific charging event to the SCF in response to the ApplyCharging operation. It has the following format:
assistrequestinstructions
The message generated by this operation causes the event that contains it to be sent automatically as the first event on a new assisting dialog.

Although you can include the correlationid parameter in the script, it is ignored and overwritten with the value received from the most recently received EstablishTemporaryConnection operation.

**callinformationreport**

This operation sends specific call information to the SCF as requested by a previous callinformationrequest operation. This operation has the following format:

```
callinformationreport [<parameters>]
```

A callinformationreport operation should have one or more of the following parameters, appearing in any order, matching the information requested in the relevant callinformationrequest operation:

```
callattemptelapsedtime <integer expression>
callstopstime <digits>
callconnectedelapsedtime <integer expression>
calleddaddress <number expression>
releasecause <cause expression>
```

The callattemptelapsedtime parameter is measured in seconds while the callconnectedelapsedtime parameter is measured in deciseconds. The callstopstime parameter is a string in the format: YYMDDHHMMSS.

**collecteduserinformation**

This operation has the following format:

```
collecteduserinformation
digits <digits>
```

This is not a distinct operation. It represents the result form of the INAP operation, promptAndCollectUserOperation. The digits parameter corresponds to the digitsResponse tag in the result.

**entityReleasedGprs**

Use this operation when the GPRS session is detached or a PDP context is disconnected and the related event is not equipped for reporting. This operation has the following format:

```
entityReleasedGprs
gprsReleaseCause <integer>
[ gprsPdPid <integer> ]
```

**entityReleasedAckGprs**

This operation has no parameters. It is the returned result for entityReleasedGprs.

**error**

An error operation has the following format:

```
error <name> [ invokeId <invoke-id> ]
```

[correlationid <digits>]
An error operation generates a U-ERROR component in the outgoing message. The name parameter determines the error code used. The following values are valid:

cancelled
cancelfailed
etcfailed
impropercallerresponse
missingcustomerrecord
missingparameter
parameteroutofrange
requestedinfoerror
systemfailure
taskrefused
unavailableresource
unexpectedcomponentsequence
unexpecteddatavalue
unexpectedparameter
unknownlegid

Some errors would typically have additional error codes, but the splpit utility supports only the ones listed here.

The invoke-id value is from the last received INVOKE component, unless you specifically define it with the invokeId parameter.

**eventreportbcsm**

This operation notifies the SCF of a call-related event that was requested by the SCF in a previous RequestReportBCSMEvent operation. Examples of call-related events would be busy or no answer. This operation has the following format:

eventreportbcsm [<event>...]

The event parameter has the following format:

eventtypebcsm <type>
[miscallinfo <miscallinfo> | monitormode <mode>]
[legid <legid> | { <integer> | ]
[eventspecificinfo <info>]

The <mode> parameter has one of the following values:

interrupted
notifyAndContinue
transparent

The <legid> parameter has one of the following values:

[sendingsideid] <legtype>
[receivingsideid] <legtype>

The <legtype> parameter has one of the following values:

l1tleg1
l1tleg2

Event specific information includes the following:

busycase <cause>
releasecause <cause>
failurecause <cause>
calledpartynumber <number>
**eventReportGprs**
This operation notifies the GSM SCF of a GPRS session or PDP context related events:

```plaintext
eventReportGprs
gprsEventType <number>
[ gprsPdPid <integer> ]
```

**eventReportAckGprs**
This operation has no parameters.

**eventReportSms**
This operation notifies the GSM service control function (gsmSCF) of a previously requested short message related event. This message has no parameters:

**informServiceCentre**
This operation is required for SMS gateway procedures between MSC and HLR. This message has the following format:

```plaintext
informServiceCentre [parameters]
```

The parameters consist of the following values:

```plaintext
storedMSISDN <number expression>
```

**initialdp**
This operation is used after a trigger detection point (TDP) to issue a request for service. This message has the following format:

```plaintext
initialdp [parameters]
```

An `initialdp` message can have any of the following parameters, specified in any order:

```plaintext
calledpartynumber <number expression>
originalcalledpartynumber <number expression>
callingpartynumber <number expression>
redirectingpartynumber <number expression>
locationnumber <number expression>
additionalcallingpartynumber <number expression>
callingpartyscategory <categoryvalue or number>
callingpartyspin <digits>
origredirreason <integer> redirindicator <integer>
eventtypebcsm <type>
appcontext <string>
extension <integer> <type> <integer> <digits>
extension <integer> <type> <integer> <digits>
idp_sccp_orig_pc <integer> // 0 - 65535
idp_sccp_orig_ssn <integer> // 0 - 255
idp_sccp_orig_tt <integer> // 0 - 255
idp_sccp_orig_np <integer> // 0 - 15
idp_sccp_orig_noa <integer> // 0 - 127
idp_sccp_orig_rti <integer> // 0 or 1
idp_sccp_orig_digits <digits>
idp_sccp_dest_pc <integer> // 0 - 65535
idp_sccp_dest_ssn <integer> // 0 - 255
idp_sccp_dest_tt <integer> // 0 - 255
idp_sccp_dest_np <integer> // 0 - 15
```
The following parameter is available for UCP handling:

AspID <string>

The following parameters are available for use with INAP CAMEL extensions:

iMSI <bcd>
countryCode <digits> networkCode <digits> locationAreaCode <integer> [ cellID <integer> ]
[ bearerCapCodingStandard <number> bearerCapITC <number> ] bearerCapITR <number> bearerCapUIProtol <number> ]
hlCharacteristicsId <number>
calledPartyBCDNumber [ [ <integer> ] *number* [ numberPlan <integer> ]
vlrNumber [ [ <integer> ] *number* [ numberPlan <integer> ]
ageOfLocationInfo <digits>
subscriberState <digits>
locationNumberLocationInfo [ [ <integer> ] *number* [ numberPlan <integer> ]
extBearerService <hex digits>
extTeleService <hex digits>
callReference <string>
[ callForwardingSSPending ]
iMEI <bcd>

The countryCode and networkCode values can be only three digits long.

Bearer capability fields are optional and are divided in two stages as shown above. If the second stage is not present, the following default values are assigned:

 bearerCapTransferMode = BC_TM_CIRCUIT (0x0), bearerCapITR = BC_ITR_64_KBIT_S (0x10) and bearerCapUIProtol = BC_UIL1_NOT_PRESENT (0xff)

The following values are available for Bearer Capability fields:

<table>
<thead>
<tr>
<th>Bearer Capability Name</th>
<th>Constants</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bearerCapCodingStandard</td>
<td>BC_CS_ITU_T</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>BC_CS_ISO_IEC</td>
<td>0x01</td>
</tr>
<tr>
<td></td>
<td>BC_CS_NATIONAL</td>
<td>0x02</td>
</tr>
<tr>
<td></td>
<td>BC_CS_NETWORK</td>
<td>0x03</td>
</tr>
<tr>
<td>bearerCapITC</td>
<td>BC_ITC_SPEECH</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>BC_ITC_UDI</td>
<td>0x08</td>
</tr>
<tr>
<td></td>
<td>BC_ITC_RDI</td>
<td>0x09</td>
</tr>
<tr>
<td></td>
<td>BC_ITC_3_1_KHZ_AUDIO</td>
<td>0x10</td>
</tr>
<tr>
<td></td>
<td>BC_ITC_UDI_TA</td>
<td>0x11</td>
</tr>
<tr>
<td></td>
<td>BC_ITC_7_KHZ_AUDIO</td>
<td>0x11</td>
</tr>
<tr>
<td></td>
<td>BC_ITC_VIDEO</td>
<td>0x18</td>
</tr>
<tr>
<td>bearerCapTransferMode</td>
<td>BC_TM_CIRCUIT</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>BC_TM_PACKET</td>
<td>0x2</td>
</tr>
<tr>
<td>bearerCapITR</td>
<td>BC_ITR_PACKET</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>BC_ITR_64_KBIT_S</td>
<td>0x10</td>
</tr>
</tbody>
</table>
The Script File

Using the slpit Utility

1-25

InitialDpGprs

When a trigger is detected at a detection point in the general GPRS state machines, this operation requests instructions from the GSM SCF. This message has the following format:

```
initialDpGprs
  gprsEventType <integer>
  gprsMsisdn <number expression>
  gprsImsi <number expression>
  gprsOriginatingReferenceNumber <number expression>
  [ gprsEndUserAddress <PdpTypeOrganisation> <PdpTypeNumber> [ <address byte> ] ]
  [ gprsRequestedQos <qos-info> ]
  [ gprsSubscribedQos <qos-info> ]
  [ gprsNegotiatedQos <qos-info> ]
  [ gprsAccessPointName <string> ]
  [ gprsChargingId <integer> ]
  [ gprsLocationInformation
    gprsmobilecountrycode <bcd>
    gprsmobilennetworkcode <bcd>
    gprsmobilolocationareacode <bcd>
    gprscellidentity <integer>
    gprsPdpinitiationtype <integer> ]
  [ gprsggsnaddress <integer> [ <integer> ] ]
  [ sgsnNumber <number expression> ]
```

The `<qos-info>` variable can have one of the following parameter values, all of which are integers:

<table>
<thead>
<tr>
<th>Bearer Capability Name</th>
<th>Constants</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_ITR_2_64_KBIT_S</td>
<td>0x11</td>
<td></td>
</tr>
<tr>
<td>BC_ITR_384_KBIT_S</td>
<td>0x13</td>
<td></td>
</tr>
<tr>
<td>BC_ITR_1536_KBIT_S</td>
<td>0x15</td>
<td></td>
</tr>
<tr>
<td>BC_ITR_1920_KBIT_S</td>
<td>0x17</td>
<td></td>
</tr>
<tr>
<td>BC_ITR_MULTIRATE</td>
<td>0x18</td>
<td></td>
</tr>
<tr>
<td>bearerCapUIProtol</td>
<td>BC_UI1_G711_U_LAW</td>
<td>0x01</td>
</tr>
<tr>
<td></td>
<td>BC_UI1_G711_A_LAW</td>
<td>0x02</td>
</tr>
<tr>
<td></td>
<td>BC_UI1_G721_32_KBIT_S</td>
<td>0x04</td>
</tr>
<tr>
<td></td>
<td>BC_UI1_H221_H242</td>
<td>0x05</td>
</tr>
<tr>
<td></td>
<td>BC_UI1_H223_H245</td>
<td>0x06</td>
</tr>
<tr>
<td></td>
<td>BC_UI1_NON_ITU_SRA</td>
<td>0x07</td>
</tr>
<tr>
<td></td>
<td>BC_UI1_ITU_V120</td>
<td>0x08</td>
</tr>
<tr>
<td></td>
<td>BC_UI1_X31_HDLC</td>
<td>0x09</td>
</tr>
<tr>
<td></td>
<td>BC_UI1_NOT_PRESENT</td>
<td>0xff</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>gprsqosprioritylevel</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>gprsqosdeloferrsdu</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>
After it detects a TDP-R, the SMS SSF uses this operation to request instructions from the GSM SCF to complete the short-message submission to the SMSC or the short message delivery to the served subscriber. This message has the following format:

```
initialDpSms
[<parameters>]
```

An `initialDpSms` message can have the following parameters in any order:

- `callingPartyNumber <number expression>`
- `destinationSubscriberNumber <number expression>`
- `idp_sccp_orig_pc <integer> // 0 - 65535`
- `idp_sccp_orig_ssm <integer> // 0 - 255`
- `idp_sccp_orig_tt <integer> // 0 - 255`
- `idp_sccp_orig_np <integer> // 0 - 15`
- `idp_sccp_orig_noa <integer> // 0 - 127`
- `idp_sccp_orig_rti <integer> // 0 or 1`
- `idp_sccp_orig_digits <digits>`
- `idp_sccp_dest_pc <integer> // 0 - 65535`
- `idp_sccp_dest_ssm <integer> // 0 - 255`
- `idp_sccp_dest_tt <integer> // 0 - 255`
- `idp_sccp_dest_np <integer> // 0 - 15`
- `idp_sccp_dest_noa <integer> // 0 - 127`
- `idp_sccp_dest_rti <integer> // 0 or 1`
- `idp_sccp_dest_digits <digits>`
- `vlrNumber [ { <integer> } ] 'number' [ numberPlan <integer> ]`
- `countryCode <digits> networkCode <digits> locationAreaCode <integer> [ cellID <integer> ]`
- `mscAddr <digits>`

**mergeCallSegmentResult**

This message has the following format with no parameters:

```
mergeCallSegmentResult
```
moForwardSm
This service forwards mobile-originated short messages between the serving mobile switching center (MSC) or the SGSN and the SMS internetworking MSC. This message has the following format:

```
moForwardSm [<parameters>]
```

For MAP version 3, this is a mobile-originated Forward-SM message, which is distinct from the mtForwardSm message. For MAP versions 1 and 2, this operation is a Forward-SM and it can originate or terminate from a mobile device, depending on the type of PDU in the SM-RP-UI.

A Forward-SM message has the following parameters in any order:

- `MapVersion <integer>`
- `SegmentedBegin`  
- `SegmentedBody`

- `imsiOA <number expression>` //optional, only valid for MAP version 3
- `privateExtension <comma separated object id string> <integer ASN.1 tag> <hex value>` // optional, only valid for MAP version 3
- `countryCode <digits> networkCode <digits> locationAreaCode <integer> cellID <integer> // optional, valid for MAP versions 2 and 3`

The `countryCode`, `networkCode`, `locationAreaCode`, and `cellID` parameters are used to construct the global cell ID. The `countryCode` and `networkCode` values can be only three digits long.

The `SM_RP_DA` field can be service center (MO) on an IMSI (MT):

- `imsi <number expression>` (optional for MAP v2/v3 segmented body)
- `lmsi <number expression>` (optional)
- `ServiceCentreAddressDA <number expression>`

In a MAP version 2 or 3 segmented MT message, the `imsi` parameter is omitted in segments after the initial segment. If it is omitted, the MT message is encoded with the `noSM_RP_DA` parameter set.

The `SM_RP_OA` field can be an MSISDN (MO) or a service center (MT):

- `msisdn <number expression>`
- `ServiceCentreAddressOA <number expression>` (optional for MAP v2/v3 segmented body)

In a MAP version 2 or 3 segmented MT message, the originating service center is omitted in segments subsequent to the initial segment. If it is omitted, the MT message is encoded with the `noSM_RP_OA` parameter set.

Depending on the MAP version, the `SM_RP_UI` field can contain one of the following PDUs:

<table>
<thead>
<tr>
<th>PDU</th>
<th>MAP version</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS-SUBMIT</td>
<td>1, 2, and 3</td>
<td></td>
</tr>
<tr>
<td>SMS-DELIVER</td>
<td>1 and 2</td>
<td>In MT-ForwardSM for version 3</td>
</tr>
<tr>
<td>SMS-STATUS-REPORT</td>
<td>2</td>
<td>In MT-ForwardSM for version 3)</td>
</tr>
</tbody>
</table>

The type of PDU is determined by the message type indicator, TP-MTI:

```
TP_MTI <number>
```
For an SMS-SUBMIT PDU, with TP-MTI=1, the following parameters are available:

TP_VPF <number>
replyPath
requestStatusReport
TP_MR <number>
TP_DA [ToN] <string>
TP_DCS <number>
TP_VP { <1 or 7 octets (numbers)>
userDataHeader { <number> <number> ... }
userDataText <string>

Note: For TP_DA, alphabetic characters (non-telephony digits) are allowed only if ToN = 5 ( alphanumeric).

For an SMS-DELIVER (TP-MTI=0), the following parameters are available:

moreMessages <0-1>
replyPath
TP_OA [ToN] <string>
TP_DCS <number>
userDataHeader { <number> <number> ... }
userDataText <string>

Note: For TP_OA, alphabetic characters (non-telephony digits) are allowed only if ToN=5 ( alphanumeric).

For an SMS-STATUS-REPORT (TP-MTI=2), the following parameters are available:

moreMessages <0-1>
TP_MR <number>
TP_RA [ToN] <string>
TP_DCS <number>
userDataHeader { <number> <number> ... }
userDataText <string>

Note: For TP_RA, alphabetic characters (non-telephony digits) are allowed only if ToN=5 ( alphanumeric).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>sccp_orig_pc</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>sccp_orig_ssn</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>sccp_orig_tt</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>sccp_orig_np</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>sccp_orig_noa</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>127</td>
</tr>
<tr>
<td>sccp_orig_rti</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>sccp_orig_digits</td>
<td>&lt;digits&gt;</td>
<td>0</td>
<td>15</td>
</tr>
</tbody>
</table>
There must be exactly one each of the imsi, lmsi, ServiceCentreAddressDA, noSM_RP_DA. There must be exactly one of msisdn, ServiceCentreAddressOA, noSM_RP_OA, imsiOA. You can use the imsiOA parameter only for MAP3.

If the SegmentedBegin parameter is present, the only other parameters allowed are MapVersion and IMSI. The result is that a TCAP_BEGIN message is sent with the appropriate application context but with no component (the User Information part of the TCAP_BEGIN message contains a MAP-OPEN with an optional IMSI in it.) If the IMSI parameter is present in the SegmentedBegin, the RP-DA in the ForwardSM should be a LMSI, but this is not enforced by the slpit utility.

If the SegmentedBody parameter is present, a normal moForwardSM operation is sent (in a Continue) but with no application context. You must always pair SegmentedBegin and SegmentedBody operations with appropriate MAP versions and a receive{} message between them.

The following segmentation scenarios are valid.

- **IMSI and LMSI parameters in a segmented message:**

<table>
<thead>
<tr>
<th>Primitve</th>
<th>MAP-OPEN</th>
<th>ForwardSM RP-DA</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCAP_BEGIN</td>
<td>imsi</td>
<td>N/A</td>
<td>Begin</td>
</tr>
<tr>
<td>TCAP_CONTINUE</td>
<td>N/A</td>
<td>lmsi</td>
<td>Body #1</td>
</tr>
<tr>
<td>TCAP_CONTINUE</td>
<td>N/A</td>
<td>noSM-RP-DA</td>
<td>Body #2</td>
</tr>
</tbody>
</table>

- **IMSI only in a segmented message:**

<table>
<thead>
<tr>
<th>Primitve</th>
<th>MAP-OPEN</th>
<th>ForwardSM RP-DA</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCAP_BEGIN</td>
<td>empty</td>
<td>N/A</td>
<td>Begin</td>
</tr>
<tr>
<td>TCAP_CONTINUE</td>
<td>N/A</td>
<td>lmsi</td>
<td>Body #1</td>
</tr>
<tr>
<td>TCAP_CONTINUE</td>
<td>N/A</td>
<td>noSM-RP-DA</td>
<td>Body #2</td>
</tr>
</tbody>
</table>

- **IMSI and LMSI in a non-segmented message:**

<table>
<thead>
<tr>
<th>Primitve</th>
<th>MAP-OPEN</th>
<th>ForwardSM RP-DA</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCAP_BEGIN</td>
<td>imsi</td>
<td>imsi</td>
<td>N/A</td>
</tr>
</tbody>
</table>

- **IMSI only in a non-segmented message:**

<table>
<thead>
<tr>
<th>Primitve</th>
<th>MAP-OPEN</th>
<th>ForwardSM RP-DA</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>sccp_dest_pc</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>sccp_dest_ssn</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>sccp_dest_tt</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>sccp_dest_np</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>sccp_dest_noa</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>127</td>
</tr>
<tr>
<td>sccp_dest_rti</td>
<td>&lt;integer&gt;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>sccp_dest_digits</td>
<td>&lt;digits&gt;</td>
<td>0</td>
<td>15</td>
</tr>
</tbody>
</table>
Call Messages

You may specify either the userDataHeader or the userDatatext or both. You must specify the header byte by byte, and in decimal or hex (with 0x as a prefix) – for example, userDataHeader {0x17 0x34}. The header is automatically prefixed with a one-byte length field.

The userDatatext parameter will be added to the packet after the userDataHeader parameter in either GSM 7-bit (default) or Unicode UCS2/UTF16 (big endian, meaning the most significant bytes in multi-byte data types are stored first) or binary, depending on the value of the data coding scheme TP_DCS.

The split utility does not support compressed user data.

You can specify TP_VF (validity period format) and TP_VP (validity period). See GSM 03.40 v7.5.0 sections 9.2.3.3 and 9.2.3.12 for encoding details. For example:

- tp_vf 0
  VPF of 0, or simply not specified, means no validity period format.

- tp_vf 1
  Enhanced format (new to MAP version 3) tp_vp {0x42 0x80 0x00 0x00 0x00 0x00 0x00}. Relative; 128 (0x80) seconds and single-shot=true.

- tp_vf 2
  Relative format tp_vf {128}, where 128 is decimal and means 645 minutes.

- tp_vpf 3
  Absolute format tp_vp {0x40 0x50 0x32 0x61 0x10 0x20 0x00}. 2004-05-23 16:01:02 GMT.

### mtForwardSM

This operation is a MAP version 3 mobile-terminated Forward-SM and is available for use with MAP. It forwards mobile terminated short messages between the gateway mobile switching center (MSC) and the servicing MSC or the SGSN.

**mtForwardSm** \[<parameters>\]

An mtForwardSm operation should have the following parameters, appearing in any order:

```
MapVersion <integer>  // only 3 is valid
SegmentedBegin
SegmentedBody
```

The SM_RP_DA field must be an IMSI number for MT-ForwardSM if the message is not getting segmented. If the message is segmented, the SM_RP_DA can be a LMSI, in which case the SegmentedBegin should contain the IMSI:

```
imsi <number expression>  // optional in the SegmentedBegin
lmsi <number expression>  // optional in the SegmentedBody
```

In a segmented MT-ForwardSM operation, the IMSI is omitted in segments following the initial segment. If it is omitted, the operation is encoded with the noSM_RP_DA parameter set.
The SM_RP_OA field must be a service center for MT-ForwardSM:

ServiceCentreAddressOA <number expression> (optional)

In a segmented MT-ForwardSM operation, the originating service center is omitted in segments following the initial segment. If it is omitted, the operation is encoded with the noSM_RP_OA parameter set.

The SM_RP_UI field must contain an SMS-Deliver or an SMS-STATUS-REPORT for MT-ForwardSM. For more information, see "moForwardSm".

```
sccp_orig_pc <integer> // 0 - 65535
sccp_orig_ssn <integer> // 0 - 255
sccp_orig_tt <integer> // 0 - 255
sccp_orig_np <integer> // 0 - 15
sccp_orig_noa <integer> // 0 - 127
sccp_orig_rti <integer> // 0 or 1
sccp_orig_digits <digits>
sccp_dest_pc <integer> // 0 - 65535
sccp_dest_ssn <integer> // 0 - 255
sccp_dest_tt <integer> // 0 - 255
sccp_dest_np <integer> // 0 - 15
sccp_dest_noa <integer> // 0 - 127
sccp_dest_rti <integer> // 0 or 1
sccp_dest_digits <digits>
```

If SegmentedBegin is present, the only other parameters allowed are MapVersion and IMSI. This results in a TCAP-BEGIN being sent with the appropriate application context but no component (the User Information part of the TC-BEGIN contains a MAP-OPEN with an optional IMSI in it). If the IMSI is present in the SegmentedBegin, the RP-DA in the ForwardSM should be a LMSI, but this is not enforced by the slpit utility.

If SegmentedBody is present, a normal moForwardSM is sent in a CONTINUE but with no application context. You are responsible for always pairing SegmentedBegin and SegmentedBody operations, with matching MapVersions and a receive {} message between them.

See the segmentation scenarios in the "moForwardSm" section for more information.

You may specify either the userData header or the text or both. You must specify the header byte by byte in decimal or hex (with a 0x prefix) – for example, userData { 0x02 0x17 0x34}. The first number in the user data header should be the length, in bytes, of the remainder of the user data header. In this case, 0x02 indicates that there are two more bytes to follow in the header.

Any user data text will be copied after the user data header, with bit padding inserted to align to a septet boundary.

**readyForSM**

This operation is available for MAP handling. It is used between the message switching center (MSC) and the VLR and between the VLR and the HLR. If a subscriber has available memory, the MSC initiates this service and the VLR indicates this condition to the HLR. If a subscriber, whose message waiting flag is active in the VLR, has radio contact in the MSC, the VLR initiates this service.

Likewise, if a subscriber has available memory, the SGSN initiates this service to indicate this to the HLR. Also, if a subscriber, whose message waiting flag is active in the SGSN has radio contact in the GPRS, the SGSN initiates this service.

readyForSM [ <parameters> ]
A `readyForSM` operation should have the following parameters, appearing in any order:

- `imsi <bcd>`
- `alertReason <alertReason> | <number>`

The `<alertReason>` value can be one of the following:

- `ms_Present`
- `memoryAvailable`

### reportSMDeliveryStatus

This operation is available for MAP handling. It is used by the message switching center (MSC) to set the message waiting data into the HLR or to inform the SLR of a successful short message (SM) transfer after polling.

`reportSMDeliveryStatus [<parameters>]`

A `reportSMDeliveryStatus` operation should have the following parameters, appearing in any order:

- `msisdn <number expression>`
- `serviceCentreAddress <number expression>`
- `smDeliveryOutcome <smDeliveryOutcome> | <number>`

The `<smDeliveryOutcome>` value can be one of the following:

- `memoryCapacityExceeded`
- `absentSubscriber`
- `successfulTransfer`

### sendRoutingInfoForSm

This operation is for MAP handling and has the following format.

`sendRoutingInfoForSm [<parameters>]`

A `sendRoutingInfoForSm` operation should have the following parameters, appearing in any order:

- `MapVersion <number>`
- `msisdn <number expression>`
- `AttemptDelivery <0-1>`
- `ServiceCentreAddress <number expression>` (optional parameters)
- `MessageTypeIndicator <number>`
- `OriginatingSmeAddr <number expression>`
- `GprsSupport <0-1>`
- `sccp_orig_pc <integer> // 0 - 65535`
- `sccp_orig_ssn <integer> // 0 - 255`
- `sccp_orig_tt <integer> // 0 - 255`
- `sccp_orig_np <integer> // 0 - 15`
- `sccp_orig_noa <integer> // 0 - 127`
- `sccp_orig_rti <integer> // 0 or 1`
- `sccp_orig_digits <digits>`
- `sccp_dest_pc <integer> // 0 - 65535`
- `sccp_dest_ssn <integer> // 0 - 255`
- `sccp_dest_tt <integer> // 0 - 255`
- `sccp_dest_np <integer> // 0 - 15`
- `sccp_dest_noa <integer> // 0 - 127`
- `sccp_dest_rti <integer> // 0 or 1`
- `sccp_dest_digits <digits>`
**sendRoutingInformation**

This operation is available for MAP handling and has the following format:

```
sendRoutingInformation [parameters]
```

A `sendRoutingInformation` operation should have the following parameters, appearing in any order:

- `interrogationType <0-1>`
- `gmscAddress <number expression>`
- `msisdn <number expression>`

(Optional parameters)

- `sccp_orig_pc <integer> // 0 - 65535`
- `sccp_orig_ssn <integer> // 0 - 255`
- `sccp_orig_tt <integer> // 0 - 255`
- `sccp_orig_np <integer> // 0 - 15`
- `sccp_orig_noa <integer> // 0 - 127`
- `sccp_orig_rti <integer> // 0 or 1`
- `sccp_orig_digits <digits>`

- `sccp_dest_pc <integer> // 0 - 65535`
- `sccp_dest_ssn <integer> // 0 - 255`
- `sccp_dest_tt <integer> // 0 - 255`
- `sccp_dest_np <integer> // 0 - 15`
- `sccp_dest_noa <integer> // 0 - 127`
- `sccp_dest_rti <integer> // 0 or 1`
- `sccp_dest_digits <digits>`

An `interrogationType` value of 0 indicates a basic call; a value of 1 indicates a forwarding call.

**smsNotification**

This operation is available for IS41 support. Your application can expect to receive one of these from an MSC when a SME comes back online after sleeping through a direct delivery attempt.

```
smsNotification [parameters]
```

An `smsNotification` operation can have the following parameters.

- `smsnot_MIN <bcd-string> // 10 digit number`
- `smsnot_ESN <integer> <integer> // 0..2^8, 0..2^24`

**smsNotificationResult**

The response to an `smsNotification` operation does not contain any parameters.

```
smsNotificationResult
```

**locationRequest**

This operation is available for IS41 support and has the following format:

```
locationRequest [parameters]
```

A `locationRequest` operation has the following parameters:

- `locreq_BID <integer> <integer> <integer> <integer> // 0..2^16, 0..2^8, 0..2^24, 0..2^8`
- `locreq_DIGITS <integer> <integer> <bcd-string>`
- `locreq_MSCID <integer> <integer> // 0..2^16, 0..2^8`
- `locreq_SYSTEMTYPECODE <integer>`
- `sccp_orig_pc <integer> // 0 - 65535`
sccp_orig_ssn <integer> // 0 - 255
sccp_orig_tt <integer> // 0 - 255
sccp_orig_np <integer> // 0 - 15
sccp_orig_noa <integer> // 0 - 127
sccp_orig_rti <integer> // 0 or 1
sccp_orig_digits <digits>
sccp_dest_pc <integer> // 0 - 65535
sccp_dest_ssn <integer> // 0 - 255
sccp_dest_tt <integer> // 0 - 255
sccp_dest_np <integer> // 0 - 15
sccp_dest_noa <integer> // 0 - 127
sccp_dest_rti <integer> // 0 or 1
sccp_dest_digits <digits>

The locreq_BID (Billing ID) parameters are: Market ID, Switch Number, ID Number, and Segment Counter. The locreq_DIGITS parameters are: Type of Digits, Nature of Number, and BCD Digits. The locreq_MSCID parameters are: Market ID, and Switch Number.

The locreq_SYSTEMMYTYPECODE parameter sets VENDOR_IDENTIFIER_, which is the only content of locreq_SYSTEMMYTYPECODE.

You can use the following values for VENDOR_IDENTIFIER:

VENDOR_IDENTIFIER_NotUsed = 0,
VENDOR_IDENTIFIER_EDS = 1,
VENDOR_IDENTIFIER_Astronet = 2,
VENDOR_IDENTIFIER_LucentTechnologies = 3,
VENDOR_IDENTIFIER_Ericsson = 4,
VENDOR_IDENTIFIER_GTE = 5,
VENDOR_IDENTIFIER_Motorola = 6,
VENDOR_IDENTIFIER_NEC = 7,
VENDOR_IDENTIFIER_Nortel = 8,
VENDOR_IDENTIFIER_NovAtel = 9,
VENDOR_IDENTIFIER_Plexsys = 10,
VENDOR_IDENTIFIER_DigitalEquipmentCorp = 11,
VENDOR_IDENTIFIER_INET = 12,
VENDOR_IDENTIFIER_Bellcore = 13,
VENDOR_IDENTIFIER_AlcatelSEL = 14,
VENDOR_IDENTIFIER_Tandem = 15,
VENDOR_IDENTIFIER_QUALCOMM = 16,
VENDOR_IDENTIFIER_Aldiscon = 17,
VENDOR_IDENTIFIER_Celcore = 18,
VENDOR_IDENTIFIER_TELOS = 19,
VENDOR_IDENTIFIER_Samnite = 20,
VENDOR_IDENTIFIER_CoralSystems = 21,
VENDOR_IDENTIFIER_SynacomTechnology = 22,
VENDOR_IDENTIFIER_DSC = 23,
VENDOR_IDENTIFIER_MCI = 24,
VENDOR_IDENTIFIER_NewNet = 25,
VENDOR_IDENTIFIER_SemaGroupTelecoms = 26,
VENDOR_IDENTIFIER_LGInformationAndCommunications = 27,
VENDOR_IDENTIFIER_CBIS = 28,
VENDOR_IDENTIFIER_Siemens = 29

locationRequestResult
This is the response to a locationRequest operation.

locationRequestResult [〈parameters〉]
A locationRequestResult operation has the following parameters:

- smsreq_ESN <integer> <integer> // 0..2^8, 0..2^24
- smsreq_MIN <integer> // must be 10 digits
- smsreq_MSCID <integer> <integer> // 0..2^16, 0..2^8

The MSCID, ESN and MIN parameters are mandatory in the response. However, if the ESN is unknown, it is set to "0 0"; if the MIN is unknown it is set to "0000000000".

**smsRequest**

This operation is available for IS41 support. Your applications should never receive one of these. It is included only to assist in testing negative cases, that is, scenarios in which the service control point, acting as an SMSC, receives an unexpected operation.

- smsRequest [<parameters>]

A smsRequest operation can have the following parameters.

- smsreq_MIN <bcd-string> // 10 digit number
- smsreq_IMSI <number expression>
- smsreq_MDN <integer> <integer> <bcd-string>
- smsreq_ESN <integer> <integer> // 0..2^8, 0..2^24
- smsreq_notificationIndicator <integer> // 0..255
- smsreq_teleserviceIdentifier <integer> // 0..65535
- sccp_orig_pc <integer> // 0 - 65535
- sccp_orig_ssn <integer> // 0 - 255
- sccp_orig_tt <integer> // 0 - 255
- sccp_orig_np <integer> // 0 - 15
- sccp_orig_noa <integer> // 0 - 127
- sccp_orig_rti <integer> // 0 or 1
- sccp_orig_digits <digits>
- sccp_dest_pc <integer> // 0 - 65535
- sccp_dest_ssn <integer> // 0 - 255
- sccp_dest_tt <integer> // 0 - 255
- sccp_dest_np <integer> // 0 - 15
- sccp_dest_noa <integer> // 0 - 127
- sccp_dest_rti <integer> // 0 or 1
- sccp_dest_digits <digits>

The smsreq_MDN parameters are: Type of Digits, Nature of Number, and BCD Digits.

**smsRequestResult**

This is the response to an smsRequest operation.

- smsRequestResult [<parameters>]

An smsRequest operation can be an Ack or a Nack. Acks contain an address, and can also return ESN data. Nacks may specify a value for the accessDeniedReason parameter.

- smsreq_ESN <integer> <integer> // 0..2^8, 0..2^24
- smsreq_address <integer> <integer> <bcd-string> // NoN, NPI, digits
- smsreq_accessDeniedReason <integer> // 0..255

**smsDeliveryPointToPoint**

This operation is available for IS41 support. You can use this to deliver a short message over IS41.
The `splt` utility supports three text-based teleservices, 4098, 4101, and 32513 and two use cases for the text message. The first is human-readable text with an optional header. The text is encoded as 7-bit ASCII for CDMA or IRA for TDMA. (The actual encoding step is independent of the `smdpp_userDataEncoding` parameter.) You may not specify both text and header for a TDMA message.

**CDMA Text Plus Header**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>0x01</td>
</tr>
<tr>
<td>length</td>
<td>The number of octets after this one.</td>
</tr>
<tr>
<td>encoding</td>
<td>The first five bits of the <code>smdpp_userDataEncoding</code> value. Note that everything after this is shifted 3 bits to the left.</td>
</tr>
<tr>
<td>num_fields</td>
<td>The number of characters (7 or 8-bit) after this octet.</td>
</tr>
<tr>
<td>header</td>
<td>Zero or more octets of GSM user-data-header. This is taken directly from the <code>smdpp_userDataHeader</code> value.</td>
</tr>
<tr>
<td>padding-1</td>
<td>Padding required to make the header end on a septet boundary. This is only done if the encoding is 2 (7-bit ASCII, default) or 3 (IA5).</td>
</tr>
<tr>
<td>text</td>
<td>Encoded message text. This will always be 7-bit ASCII.</td>
</tr>
<tr>
<td>padding-2</td>
<td>Padding required to make this whole block end on an octet boundary.</td>
</tr>
</tbody>
</table>

**TDMA text**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>The number of octets following this one.</td>
</tr>
<tr>
<td>type</td>
<td>The first 5 bits of <code>smdpp_userDataEncoding</code>.</td>
</tr>
<tr>
<td>padding</td>
<td>3 bits of padding, so that text starts on an octet boundary.</td>
</tr>
<tr>
<td>text</td>
<td>Text as something resembling IRA, with each character 7 bits wide but stored in an octet with the high bit off.</td>
</tr>
</tbody>
</table>

The second scenario does something a little more exotic with the text. In this case, you cannot use the human-readable text parameter (`smdpp_userDataText`) – you must put the raw bytes of the message into the header (`smdpp_userDataHeader`). Here’s how the header and text are packed into the message for the different use cases:
CDMA Header Only

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>length encoding num_fields data padding</td>
</tr>
</tbody>
</table>

These fields have the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>length</td>
<td></td>
</tr>
<tr>
<td>encoding</td>
<td></td>
</tr>
<tr>
<td>num_fields</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>The octets specified in smdpp_userDataHeader shifted 3 bits to the left.</td>
</tr>
<tr>
<td>padding</td>
<td>Empty bits required to bring the block to an octet boundary.</td>
</tr>
</tbody>
</table>

TDMA Header

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>type padding data</td>
</tr>
</tbody>
</table>

These fields have the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td></td>
</tr>
<tr>
<td>padding</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>The octets specified in the smdpp_userDataHeader</td>
</tr>
</tbody>
</table>

Note that the translation of text from the human-readable input form to ASCII or IRA is not perfect. When in doubt, try using the header to set the raw data.

A smsDeliveryPointToPoint may have the following parameters. The smdpp_teleservice parameter is mandatory, and must be set according to IS-41-D before the TDMA or CDMA sections can be used. For more information about these parameters, please consult TIA/EIA-41-D-1997 (IS41), 3GPP2 C.S0015-A (CDMA) and TIA/EIA-136-710-C (TDMA).

Each teleservice may place a particular restriction on the data specified. These restrictions aren’t generally enforced by the slpit utility, because one may want to send bad data. The following are common restrictions:

- **CDMA 4098**
  - No userdata header present.

- **CDMA 4100**
  - Encoding type is 0.

- **TDMA 32513**
  - No userdata header present.

These are the SMDPP parameters:
### Parameter Encoding Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>smdpp_teleservice</td>
<td>&lt;integer&gt;</td>
<td>4098, 4100, 32513</td>
</tr>
<tr>
<td>smdpp_MIN</td>
<td>&lt;bcd-string&gt;</td>
<td>10 digit</td>
</tr>
<tr>
<td>smdpp_ESN</td>
<td>&lt;integer&gt;&lt; integer&gt;</td>
<td>0...2⁸, 0...2²⁴</td>
</tr>
<tr>
<td>smdpp_origAddr</td>
<td>&lt;integer&gt;&lt; integer&gt;&lt;bcd-string&gt;</td>
<td>NoN, NPI, digits</td>
</tr>
<tr>
<td>smdpp_origOrigAddr</td>
<td>&lt;integer&gt;&lt; integer&gt;&lt;bcd-string&gt;</td>
<td>NoN, NPI, digits</td>
</tr>
<tr>
<td>smdpp_destAddr</td>
<td>&lt;integer&gt;&lt; integer&gt;&lt;bcd-string&gt;</td>
<td>NoN, NPI, digits</td>
</tr>
<tr>
<td>smdpp_origDestAddr</td>
<td>&lt;integer&gt;&lt; integer&gt;&lt;bcd-string&gt;</td>
<td>NoN, NPI, digits</td>
</tr>
<tr>
<td>smdpp_messageCount</td>
<td>&lt;integer&gt;</td>
<td>0...2⁸</td>
</tr>
<tr>
<td>smdpp_notInd</td>
<td>&lt;integer&gt;</td>
<td>0...2⁸</td>
</tr>
<tr>
<td>smdpp_chargeInd</td>
<td>&lt;integer&gt;</td>
<td>0...2⁸</td>
</tr>
<tr>
<td>smdpp_userDataEncoding</td>
<td>&lt;number&gt;</td>
<td></td>
</tr>
<tr>
<td>smdpp_userDataHeader</td>
<td>{&lt;number&gt;...}</td>
<td></td>
</tr>
<tr>
<td>smdpp_userDataText</td>
<td>&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td>CDMA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDMA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following lists show the common encoding values:

- **CDMA**

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>octet-unspecified</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>Extended protocol message</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>7-bit ASCII (default)</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>IA5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>UNICODE</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Shift JIS</td>
<td>8 / 16</td>
</tr>
<tr>
<td>6</td>
<td>Korean</td>
<td>8 / 16</td>
</tr>
<tr>
<td>7</td>
<td>Latin/Hebrew</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>Latin</td>
<td>8</td>
</tr>
</tbody>
</table>

- **TDMA**

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IRA</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>User specific</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Latin</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>Latin/Hebrew</td>
<td>8</td>
</tr>
</tbody>
</table>
If you are using the CDMA teleservices, you can specify the following parameters in the CDMA subsection:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>smdpp_messageId</td>
<td>&lt;integer&gt;&lt;integer&gt;&lt;boolean&gt;</td>
<td>0..2^4, 0..2^16, true/false</td>
</tr>
<tr>
<td>smdpp_validityPeriod</td>
<td>&lt;integer&gt;</td>
<td>See 4.5.6.1 of 3GPP2 CS 15-A</td>
</tr>
<tr>
<td>smdpp_validityPeriod</td>
<td>&lt;string&gt;</td>
<td>YYMMDDhhmms</td>
</tr>
<tr>
<td>smdpp_deferredDeliveryTime</td>
<td>&lt;integer&gt;</td>
<td>See 4.5.6.1 of 3GPP2 CS 15-A</td>
</tr>
<tr>
<td>smdpp_deferredDeliveryTime</td>
<td>&lt;string&gt;</td>
<td>YYMMDDhhmmss</td>
</tr>
<tr>
<td>smdpp_priorityInd</td>
<td>&lt;integer&gt;</td>
<td>0..3</td>
</tr>
<tr>
<td>smdpp_privacyInd</td>
<td>&lt;integer&gt;</td>
<td>0..3</td>
</tr>
<tr>
<td>smdpp_languageInd</td>
<td>&lt;integer&gt;</td>
<td>0..255</td>
</tr>
<tr>
<td>smdpp_alertOnDelivery</td>
<td>&lt;integer&gt;</td>
<td>0..3</td>
</tr>
<tr>
<td>smdpp_DAKRequested</td>
<td>&lt;boolean&gt;</td>
<td>true/false</td>
</tr>
<tr>
<td>smdpp_MAKRequested</td>
<td>&lt;boolean&gt;</td>
<td>true/false</td>
</tr>
<tr>
<td>smdpp_RAKRequested</td>
<td>&lt;boolean&gt;</td>
<td>true/false</td>
</tr>
</tbody>
</table>

If you are using the TDMA teleservice, you can specify the following parameters in the TDMA subsection:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>smdpp_messageTypeInd</td>
<td>&lt;integer&gt;</td>
<td>0..2^3</td>
</tr>
<tr>
<td>smdpp_messageRef</td>
<td>&lt;integer&gt;</td>
<td>0..2^13</td>
</tr>
<tr>
<td>smdpp_privacyInd</td>
<td>&lt;integer&gt;</td>
<td>0..2^3</td>
</tr>
<tr>
<td>smdpp_urgencyInd</td>
<td>&lt;integer&gt;</td>
<td>0..2^2</td>
</tr>
<tr>
<td>smdpp_DAKRequested</td>
<td>&lt;boolean&gt;</td>
<td>true/false</td>
</tr>
<tr>
<td>smdpp_MAKRequested</td>
<td>&lt;boolean&gt;</td>
<td>true/false</td>
</tr>
<tr>
<td>smdpp_messageUpdating</td>
<td>&lt;boolean&gt;</td>
<td>true/false</td>
</tr>
<tr>
<td>smdpp_vp_absolute</td>
<td>&lt;integer&gt;</td>
<td>0..1</td>
</tr>
<tr>
<td>smdpp_vp_relativeTimerValue</td>
<td>&lt;integer&gt;</td>
<td>0..255</td>
</tr>
<tr>
<td>smdpp_vp_absoluteSeconds</td>
<td>&lt;integer&gt;</td>
<td>0..2^32</td>
</tr>
<tr>
<td>smdpp_vp_absoluteTZOffsetDirection</td>
<td>&lt;integer&gt;</td>
<td>0..1</td>
</tr>
<tr>
<td>smdpp_vp_absoluteTZOffsetMinutes</td>
<td>&lt;integer&gt;</td>
<td>0..720</td>
</tr>
<tr>
<td>smdpp_vp_absoluteTZOffsetDSI</td>
<td>&lt;integer&gt;</td>
<td>0..1</td>
</tr>
</tbody>
</table>

**smsDeliveryPointToResult**

This is the response to an smsDeliveryPointToPoint operation.

```text
smsDeliveryPointToPointResult [parameters]
```
An `smsDeliveryPointToPoint` operation may be an Ack or a Nack. Nacks contain an `SMS_CauseCode` parameter, specified as follows:

```plaintext
smdpp_causeCode <integer> // 0..2^8
```

### `splitLegResult`

This parameter is available for INAP level 2 (CS-2) handling.

```plaintext
splitLegResult
```

A `splitLegResult` operation has no parameters.

### `specializedresourceReport`

```plaintext
specializedresourceReport
```

This operation has no parameters.

### `tcapReject`

```plaintext
tcapReject
  problemtype <type>
  generalproblem <problem>
```

A `tcapReject` operation will have the `split` utility send a TCAP_REJECT primitive on the main dialog. The problem type and ID are taken from the parameters. The mandatory parameter is `problemtype` which must be an integer from the following list:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>none</td>
</tr>
<tr>
<td>0</td>
<td>general</td>
</tr>
<tr>
<td>1</td>
<td>invoke</td>
</tr>
<tr>
<td>2</td>
<td>return_result</td>
</tr>
<tr>
<td>3</td>
<td>return_error</td>
</tr>
</tbody>
</table>

The `generalproblem` parameter is also an integer, from 0 to 255. The reject source is set to TCAP_REJECT_LOCAL.

### `unstructuredSS`

This operation is available for MAP handling. It sends a `MAP2_ProcessUnstructuredSSRequest`. The only language available is the default GSM alphabet because that’s the only language that TC_PROTOS currently supports.

```plaintext
unstructuredSS [<parameters>]
```

An `unstructuredSS` operation can have the following parameters, appearing in any order:

- `msisdn <number expression>`
- `msisdnReference <number expression>`
- `originatingReference <number expression>`
- `destinationReference <number expression>`
- `ussd <string>`
- `iMEI <bcd>`
- `countryCode <digits>`
- `networkCode <digits>`
- `locationAreaCode <integer>`
The Script File

Using the slpit Utility

1-41

cellID <integer>
scpp_orig_pc <integer> // 0 - 65535
scpp_orig_ssn <integer> // 0 - 255
scpp_orig_tt <integer> // 0 - 255
scpp_orig_np <integer> // 0 - 15
scpp_orig_noa <integer> // 0 - 127
scpp_orig_rti <integer> // 0 or 1
scpp_orig_digits <digits>
scpp_dest_pc <integer> // 0 - 65535
scpp_dest_ssn <integer> // 0 - 255
scpp_dest_tt <integer> // 0 - 255
scpp_dest_np <integer> // 0 - 15
scpp_dest_noa <integer> // 0 - 127
scpp_dest_rti <integer> // 0 or 1
scpp_dest_digits <digits>

The countryCode and networkCode can be only three digits long.

A MapOpen is inserted into the TCAP primitive’s UserInformation area. The
msisdReference is used to populate the msisdReference in the MapOpen. The
destinationReference populates the destinationReference in the MapOpen. The
originatingReference is used to populate originationReference in the MapOpen. The
msisd is used to populate the msisd parameter in the UnstructuredSSRequest.

[empty]
The slpit utility can also send empty TCAP primitives. You can accomplish this by
leaving the body of the send message blank as shown in the following example:

send
{
}

Receive Message Operations

The following operations are available in the receive message portion of a call
definition.

abort
The format of this parameter is simply:

abort

Use this operation to receive aborts that you expect. The Calls Aborted count is not
updated, but the Calls Succeeded count is updated. In this case, the script is expecting
an abort when so the test is successful in that it received one and continues. This is
especially useful when running multiple calls because if it treated it like a standard
abort, slpit would stop processing.

anyTimeInterrogation
This operation is available for handling MAP. It is used for time information enquiries
between GSM SCF and HLR. It has the following format and parameters:

anyTimeInterrogation
    [ locationInformation
        [ age <integer comparator>
        ]
        [ geographical <number comparator>
        ]
        [ vrl <number comparator>
        ]
You can define the parameters to the locationInformation part of this operation in any order.

**applycharging**

This operation has the following format:

```plaintext
applycharging
thresholdtime <integer comparator>
[warnings <integer comparator>]
```

The following format is available for handling CAMEL:

```plaintext
applyCharging
maxDuration <integer>
[release <integer> tone <integer>]
[tariff <integer>]
```

**applyChargingGprs**

This operation is available for GPRS handling and it takes one of the following two forms:

```plaintext
applyChargingGprs
gprsTransferredVolume <integer>
[gprsTariffSwitchInterval <integer>]
[gprsPdPid <integer>]
```

```plaintext
applyChargingGprs
gprsElapsedTime <integer>
[gprsTariffSwitchInterval <integer>]
[gprsPdPid <integer>]
```

**callinformationrequest**

This operation requests the SSF to record information about a call and use the CallInformationReport operation to report it to the SCF using the CallInformationReport operation has the following format:

```plaintext
callinformationrequest <requested fields>
```

A callinformationrequest operation must have at least one of the following labels requesting particular information. They may appear in any order:

```plaintext
callattemptelapsedtime
callstopetime
callconnectedelapsedtime
calledaddress
releasecause
```

These are effectively flags and do not have any associated values.

---

**Note:** The received operation check ensures that all expected types are present but not for extra types that were not expected.
**collectinformation**

This operation requests the SSF to perform the call processing actions that collect destination information from a calling party. This operation has the following format, with no parameters.

`collectinformation`

**connect**

This operation requests the SSF to route a call to its destination. It has the following format:

`connect [parameters]`

The `connect` operation can have any of the following parameters in any order:

- `originalcalledpartnumber <number comparator>`
- `callingpartynumber <number comparator>`
- `redirectingpartynumber <number comparator>`
- `destroutingaddr <number comparator>`
- `callingpartyscategory <integer> | callingpartyscategory <category>`
- `genericnumbers numberqualifier <qualifier> <number comparator>`
- `...numberqualifier <qualifier> <number comparator>`

At the least, you must include the `destroutingaddr` parameter. The `<category>` parameter is one of:

- `unknowncategory`
- `operatorfrench`
- `operatorenglish`
- `operatorgerman`
- `operatorrussian`
- `operatorspanish`
- `ordinarycallingsubscriber`
- `callingsubscriberwithpriority`
- `datacall`
- `testcall`
- `payphone`

The `<qualifier>` parameter is either an integer value or one of the following:

- `additionalCalledNumber`
- `additionalConnectedNumber`
- `additionalCallingNumber`
- `additionalOriginalCalledNumber`
- `additionalRedirectingNumber`
- `additionalRedirectingNumber`

**connectGprs**

This operation is available for GPRS handling. When establishing a PDP context, it modifies the Access Point Name. This operation has the following format:

```
cconnectGprs
gprsAccessPointName <string>
[ gprsPdPid <integer> ]
```
connecttoresource
On receipt from the GSM SCF, this operation connects the IP to the incoming call. This operation has the following format:

```plaintext
connecttoresource
  [ legid <integer> ]
  [ address <digits> ]
```

If no address is specified, the received operation must have `none` indicated for its `resourceAddress` tag. If an address is specified, it must match the address in the `ipRoutingAddress` tag.

continue
This operation requests the SSF to proceed with call processing at the detection point (DP) where it previously suspended call processing to wait for instructions from the SCF. This operation has the following format and no parameters:

```plaintext
continue
```

continueGprs
This operation is available for GPRS handling. It requests the GPRS SSF to proceed with the GPRS session or context processing at the detection point (DP) where it previously suspended processing to wait for instructions from the GSM SCF. It has the following format:

```plaintext
continueGprs [ gprsPdPid <integer> ]
```

The release cause must be between 0 and 255 inclusive.

continueSms
This operation requests the SMS SSF to proceed with processing at the detection point (DP) where it previously suspended processing to wait for instructions from the GSM SCF. It is available for CAMEL handling and it has no parameters.

```plaintext
continueSms
```

continuewithargument
Requests the GSM SSF to proceed with call processing at the detection point (DP) at which it previously suspended call processing to wait for instructions from the GSM service control function. It also provides additional service-related information to the called party or the calling party while call processing proceeds.

Parameters that are provided in the operation replace the corresponding signalling parameters in the call control function (CCF) and are used in subsequent call processing. Parameters that are not replaced by the operation retain their value in the CCF for subsequent call processing. This operation is available for INAP level 2 (CS-2) handling and has the following format:

```plaintext
continuewithargument legid <integer>
```

The `legid` parameter is the only supported parameter and it is required.

disconnectforwardconnection
This operation is used in two cases: 1) To disconnect a connection to a specialized resource function (SRF) and 2) to clear a connection to an assisting SSF. In the first
case, it disconnects a forward connection from the SSF. In the second case, it
disconnects the temporary connection between the initiating SSF and the assisting SSF
and between the assisting SSF and its associated SRF. The operation has the following
format with no parameters:
disconnectforwardconnection

disconnectforwardconnectionwithargument
This operation is available for INAP level 2 (CS-2) handling.
disconnectforwardconnectionwithargument partytodisconnect legid <integer>
The partytodisconnect parameter with the legid variant is the only supported
parameter for this operation.

disconnectleg
This operation is available for INAP level 2 (CS-2) handling. It requests the GSM SSF
to release a leg associated with the call. Other legs are retained.
disconnectleg reason <cause comparator> [legid <legid> | ( <integer> )]
Please see the description of "releasecall" for the possible values for the reason
parameter. See the description of "eventreportbcsm" for a description of the possible
values for legid.

establishtemporaryconnection
This operation creates a connection to a resource for a limited period of time to play an
announcement or collect information and so on. It has the following format and
parameter:
establishtemporaryconnection address <digits>
The address parameter, which is a string in double quotes, is mandatory.

furnishcharginginformation
This operation requests the SSF to generate or register a call record or to include some
information in the default call record. This operation has no parameters:
furnishcharginginformation

Note: Increasing the output level causes the parameters of the
received operation to be written out even though the splt utility does
not check them.

mergecallsegments
This operation is available for INAP level 2 (CS-2) handling. It has the following
format and mandatory parameters
mergecallsegments sourcecallsegment <integer> targetcallsegment <integer>

moForwardSmResult
This operation is available for MAP handling. It has no parameters:
moforwardresults

**mtForwardSmResult**
This operation is available for MAP handling. It has no parameters:

```
mtForwardSmResult
```

**playannouncement**
This operation is used for inband interaction with an analog user or for interaction with an Integrated Services Digital Network (ISDN) user. It has the following format and parameters:

```
playannouncement
    [connectedparty <integer>]
    annid <integer comparator> [, <integer comparator>...]  
    [variableparts <parts>]
```

If you include the `variableparts` parameter, you must include one or more of the part value specifiers:

```
price <integer> <integer>  
digits <digits>  
time <integer> <integer>  
date <integer> <integer> <integer>  
integer <integer>
```

These correspond to the obvious sub-tags in the `variableparts` parameter of the outgoing operation. The `price` specifier should have two integer parameters giving dollars (or big currency unit) and cents (or little currency unit) respectively. You specify the `time` value in hours and minutes and the `date` value as day of the month, month number (1-12), and year (0-99).

**prearrangedend**
This operation has no parameters. It expects the other side of the dialog to send the fake TCAP primitive `TCAP_PRE_END`.

```
prearrangedend
```

**promptandcollectuserinformation**
This operation interacts with a user to collect information. It has the following format and parameters:

```
promptandcollectuserinformation
    annid <integer comparator> [, <integer comparator>...]  
    [minnumberofdigits <integer>]
    maxnumberofdigits <integer>
    [<digit parameter>...]
    [variableparts <part> [<part>...]]
```

You can use the following specifiers for `digit parameter` in any order:

```
endofreplydigit <digits>  
canceldigit <digits>  
startdigit <digits>  
interdigittimeout <integer>  
firstdigittimeout <integer>
```

If you include the `variableparts` parameter, you must specify one or more of the following part value specifiers. See the description of "playannouncement" for more information.
readyForSMResult
This operation is available for MAP handling. It has no parameters:

readyForSMResult

realsecall
This operation causes the SCF to terminate an existing call at any phase for all parties. It has the following format and one parameter:

realsecall reason <cause comparator>

You can specify an integer for the reason parameter or one of the following names:

unalloc_num
norm_call_clr
user_busy
no_user_resp
no_answer
call_rejected
num_changed
out_of_order
inval_num_fmt
normal
temp_failure

releaseGprs
This operation causes the GSM service control function (SCF) to terminate an existing GPRS Session or PDP context at any phase. It is available for GPRS handling. It takes one of the following two forms:

releaseGprs gprsReleaseCause <integer> [ gprsPdPid <integer> ]

releaseSms
This operation causes the GSM (SCF) to terminate a short message submission attempt or short message delivery attempt and is allowed only within a control relationship. It is available for CAMEL handling and has the following parameter:

releaseSms reason <integer comparator>

reportSMSDeliveryStatusResult
This operation is available for MAP handling:

reportSMSDeliveryStatusResult [ <parameters> ]

A reportSMSDeliveryStatusResult operation must have the following parameter:

msisdn <number expression>

requestreportbcsmevent
This operation causes the SSF to monitor for call-related BCSM events such as busy or no-answer and notify the SCF when one is detected. This operation has the following format:

requestreportbcsmevent [ <event> ... ]

A requestreportbcsmevent must have one or more event descriptions:

eventtypebcsm <type> [ monitormode <mode> ] [ legid <legid> | ( <integer> ) ]
The value of legid must be one of the following:

[sendingsideid] <legtype>
[receivingsideid] <legtype>

The value of criteria must be one of the following:

numberofdigits <integer comparator>
applicationtimer <integer comparator>

For example, you can have:

eventTypeBCSM oNoAnswer (2) dpspecificcriteria applicationTimer 20

**requestReportGprsEvent**

Causes the GSM SCF to request the GPRS SSF to monitor for a GPRS session event or a PDP context event, such as establish or detach, and to notify the GSM SCF when one is detected. You can request monitoring of more than one event in a single operation but each one will be reported in a separate EventReportGPRS operation. This operation has the following format:

RequestReportGprsEvent gprsEventType <number> [ gprsPdPid <integer> ]

**requestReportSmsEvent**

Causes the GSM service control function to request the SMS SSF to monitor for a short message related event such as failure, delivery, or submission, and notify the GSM SCF when it detects one. You can request monitoring of more than one event with a single operation but each event will be reported in a separate EventReportSMS operation. This operation is available for CAMEL handling and has the following format.

eventTypeSms [ smsFailure | smsSubmitted ]

**resettimer**

This operation causes the SCF to refresh the tSSF application timer to avoid the tSSF time-out at the service SSF. This operation supports only one parameter, timervalue. You cannot specify the timer ID and it is not checked in the received operation. It defaults to tSSF.

resettimer timervalue <integer>

**returnError**

The operation has the following format:

returnError|tcapError [invokeID <integer>] errorCode <integer>

The returnError operation and the tcapError parameter operation are synonymous. If you specify invokeID, it must match the response. Otherwise, the returned invokeID is not checked. In the slpit script, calls start with an invokeID of 0 and the value is increased by 1 for each subsequent call.

**sendcharginginformation**

This operation instructs the SSF on the charging information to be sent. No parameters are supported for this operation and the received content is not validated.
sendcharginginformation

**sendRoutingInfoForSmResult**
This operation is available for MAP handling and has the following format and parameters:
```
sendRoutingInfoForSmResult [ imsi <number comparitor> ] [ nnn <number comparitor>]
```

**sendRoutingInformationResult**
This operation is available for MAP handling and has the following format and parameters:
```
sendRoutingInformationResult [ imsi <number comparitor> ]
[ nnn <number comparitor> ]
```

**splitleg**
Causes the GSM SCF to request the GSM SSF to separate one party from the source call segment and place it in a new target call segment. This operation is available for INAP level 2 (CS-2) handling and has the following format and parameters.
```
splitleg legtobesplit <integer> newcallsegment <integer>
```

**tcapError**
This operation has the following format and parameters:
```
returnError|tcapError [invokeID <integer>] errorCode <integer>
```

The `returnError` parameter and `tcapError` parameter are synonymous. If you specify `invokeID`, it must match the response. Otherwise, the returned `invokeID` is not checked. In the `slpit` script, calls start with an `invokeID` of 0 and the value is increased by 1 for each subsequent call.

**unstructuredSSResult**
This operation is available for MAP handling and has the following format and parameters.
```
unstructuredSSResult ussdString <string>
```

The `ussdString` parameter is the expected parameter in an `unstructuredSSResult` operation. If you do not specify it, no check is performed. If a check is performed, the string must match the returned string, or the call will be counted in the failed call statistics.
Example Scripts

This section illustrates the call definition statements that you could find in a split script file for two sample calls: a standard point A to point B call and a call that plays an announcement.

A Standard call

The statements in this example define a standard point A to point B call with one subsequent reservation in which the called party hangs up. This is not a CAMEL call.

```plaintext
define call atb_two_periods {
  DN ?= "39421234567"
  CLI ?= "3099440000"

  send {
    initialdp
    calledpartynumber DN
    callingpartynumber CLI
    callingpartyscategory 10
    locationnumber CLI
    eventtypebcsm analyzedinformation
  }

  receive {
    applycharging
    thresholdtime any -> threshold
    warningtime threshold - 10

    requestreportbcsmevent
    eventtypebcsm omidcall (2)

    requestreportbcsmevent
    eventtypebcsm oCalledPartyBusy (2)
    eventtypebcsm oNoanswer (2)
    eventtypebcsm oabandon (1)
    eventtypebcsm RouteselectFailure
    eventtypebcsm oDisconnect (2)
    eventtypebcsm oDisconnect (1)

    callinformationrequest
    callattemptElapsedTime
    callstopTime
    callConnectedElapsedTime
    calledaddress
    releaseCause

    connect
    destroutingaddr DN
  }

  talktime = threshold

  wait 1.0

  send {
    eventreportbcsm
    eventtypebcsm omidcall (2)
  }
```
receive {
  applyCharging
    thresholdtime any -> threshold
    warningtime threshold - 10
}

RequestReportBCsMEvent
  eventtypebcsm omidcall (2)
  continue
}

talktime = talktime + threshold

wait 1.0

send {
  eventreportbcsm
    eventtypebcsm odisconnect (2)

  applychargingreport
    thresholdtime 20
    endofcallindicator 1

  callinformationreport
    callattemptelapsedtime 10
    callstopetime "001002000000"
    callConnectedElapsedTime (talktime - 20) * 10
    calledaddress DN
    releasecause 31
}

receive {
  releasecall
    reason 31
}

startcall atb_two_periods using once

A Call that Plays an Announcement

The statements in this example define a simple call that requests assistance and plays an announcement.

define call assisting_ip_pa {
  SERVICE_NUMBER = "555801"

  send {
    initialdp
      calledpartynumber SERVICE_NUMBER
      callingpartynumber "40002000"
      callingpartyscategory 10
      locationnumber "40002000"
      eventtypebcsm analyzedinformation
  }

  receive {
    establishtemporaryconnection
      address "1234"
  }
}
send {
  assistrequestinstructions
}

receive {
  playannouncement
    annid any
}

// Might receive the abort any time after sending the SRR.
allow abort assisting

send {
  specializedresourcereport
}

receive {
  disconnectForwardConnection
  releasecall
    reason normal
}

// The abort might not have arrived yet.
  abort open
The SS7 protocol suite is a set of telephony signaling protocols that are used to establish and terminate telephone calls on public switched telephone networks. The SS7 protocol suite provides additional services as well, including number translation, local number portability, prepaid billing mechanisms, short message service (SMS), and a variety of other services.

Each protocol within a suite usually has a particular purpose. Such modularization makes design and assessment of the protocols easier. Because each protocol module usually communicates with two others, they are ordinarily considered as layers in a protocol stack. The lowest-layer protocol performs the low-level, physical interaction with the network hardware. Higher layers add more features.

Figure A–1 illustrates the SS7 protocol suite:

The INAP Protocol

The INAP protocol is the signalling protocol that is used in Intelligent Networks (INs). INAP was developed by the International Telecommunications Union (ITU), and is recognized as an international standard. The functionality of INAP has been defined and implemented by the ITU in segments called capability sets. The first version was Capability Set 1 (CS-1) and Capability Set 2 (CS-2) is currently available.

INAP communicates between a service switching point (SSP), network media resources (intelligent peripherals), and a centralized network database called a service control point (SCP). The SCP encompasses operator or third-party-derived service logic programs and data.
The CAP Protocol

The CAMEL Application Part (CAP) protocol is a signalling protocol in the IN architecture and is layered on top of the TCAP protocol. It makes possible the implementation of carrier-grade, value added services like unified messaging services, prepaid services, fraud control, and Freephone (800 number calls) in both the Global System for Mobile Communication (GSM) voice and General Packet Radio Service (GPRS) data networks. CAMEL is a means of adding intelligent applications to mobile networks. It builds upon established practices in the fixed-line telephony business that are generally considered part of the INAP CS-2 protocol.

The MAP Protocol

The Mobile Application Part (MAP) protocol supplies an application layer for nodes in the following networks:

- GSM (mobile) networks
- Universal Mobile Telecommunications System (UMTS) networks
- GPRS networks

The nodes in these networks use the MAP protocol to communicate with each other so they can provide services to mobile phone users. These services include mobility services such as location management to support roaming, call handling, SMS for text messaging, packet data protocol (PDP) services for GPRS, and operation and maintenance, as well as other services.

The IS-41 Protocol

The Interim Standard 41 protocol enables mobile, cellular telecommunications operations between different networks. It is similar to GSM and supports capabilities such as handover between networks, roaming authentication, and SMS delivery. It includes the Visitor Location Register (VLR) and Home Location Register (HLR) databases.

The TCAP Protocol

The TCAP protocol provides a presentation layer that facilitates the distribution of intelligent network services. The presentation layer deals with data format, operating system compatibility, and encapsulating data to send over the network. Fundamentally, TCAP simplifies simultaneous communications between subsystems on the same machines by using transaction IDs to associate multiple messages with a particular transaction.

In intelligent networks TCAP transports INAP and in mobile phone networks it transports MAP. See "Understanding TCAP primitives" for more information.

The SCCP Protocol

The Signaling Connection Control Part (SCCP) protocol is a routing protocol that routes TCAP messages to their proper database. SCCP provides connectionless and connection-oriented network services. SCCP provides subsystem numbers that enable messages to be addressed to specific applications or subsystems at signaling points. SCCP is the transport layer for TCAP-based services such as calling card, local number portability, wireless roaming, personal communications services (PCS), and freephone (800 numbers).
The M3UA Protocol

M3UA stands for Message Transfer Part Level 3 (MTP3) User Adaptation Layer. The M3UA protocol enables the SS7 protocol User Part SCCP, as well as others, to run over internet protocol instead of telephony equipment. The M3UA protocol is generally transmitted by using the services of Stream Control Transmission Protocol (SCTP).

The SUA Protocol

SUA stands for the SCCP User Adaptation layer. The SUA protocol facilitates the transfer of SCCP user messages, such as TCAP, between the signalling gateway and the application server process (ASP).

The SCTP Protocol

The Stream Control Transmission Protocol (SCTP) is a transport-layer protocol that delivers in-sequence messages. It performs path selection and provides fail-over support for duplicated paths in the network.

The SCTP protocol was originally designed to transport telephony over the internet, but it has evolved to have other purposes as well.

The Internet Protocol

The Internet Protocol (IP) provides routing for data packets from source to destination hosts based on IP addresses. It facilitates the internetworking that constitutes the internet and defines structures that enclose data and add the source and destination addresses. Because it is often used together with the Transport Control Protocol, it is frequently referred to as TCP/IP. It runs on top of data link interfaces such as Ethernet and Wi-Fi, operating at layer 3 of the OSI model, which is the network layer. The network layer provides routing and switching functionality to transmit data between nodes.