doubango 3GPP IMS/LTE Framework v1.0.0

Programmer's Guide

by

Mamadou Diop
diopmamadou {AT} doubango.org

License

Table of Contents

1 Foreword

3GPP IMS (IP Multimedia Subsystem) is the next generation network for delivering IP multimedia services. IMS is standardized by the 3rd Generation Partnership Project (3GPP). IMS services could be used over any type of network, such as 3GPP LTE, GPRS, Wireless LAN, CDMA2000 or fixed line.

doubango is an experimental, open source 3GPP IMS/LTE framework for both embedded and desktop systems. The framework is written in ANSI-C to ease portability and has been carefully designed to efficiently work on embedded systems with limited memory and low computing power and to be extremely portable.

doubango is a set of small APIs putted together to form a powerful 3GPP IMS/LTE framework. These stacks implement many protocols such as SIP/SDP, HTTP/HTTPS, DNS, DHCPv4/DHCPv6, SigComp, MSRP, XCAP, NAT Traversal (STUN, TURN and ICE) …

2 Scope

This user guide is a reference document explaining how to develop IMS applications using doubango.

3 Definitions and abbreviations

3.1 Definitions

- Well-known C object
- SIP session is any …
- SIP action is any … and performed on a SIP session
- Media Session is any … (Audio, Video, MSRP …)

3.2 Abbreviations

3GPP

LTE

IMS

IANA

SM

SC

SM-RL

SM-TL

RPDU

TPDU

PDU

4 General introduction

4.1 Architecture

4.2 IANA

doubango has been assigned private enterprise number (PEN) **35368** by IANA. The complete list is available at http://www.iana.org/assignments/enterprise-numbers. All network protocols (e.g. DHCPv6) will use this number in the outgoing requests. If you would like to use your own enterprise number, then you should modify `TNET_IANA_PEN` value (*tinyNET_config.h* from tinyNET project).

4.3 Projects

4.3.1 tinySAK

Prefix: `tsk`
API reference: http://doubango.org/API/tinySAK/

4.3.2 tinySigComp

Prefix: `tcomp`
API reference: http://doubango.org/API/tinySigComp/

4.3.3 tinySMS

Prefix: `tsms`

API reference: http://doubango.org/API/tinySMS/

4.3.4 tinyNET

You MUST call `tnet_startup()` before using any network function prefixed with `tnet_`. `tnet_cleanup` is used to terminate use of network functions.

Prefix: `tnet`
API reference: http://doubango.org/API/tinyNET/

4.3.5 tinyIPSec

Prefix: `tipsec`
API reference: http://doubango.org/API/tinyIPSec/

4.3.6 tinyHTTP

Prefix: `thttp`
API reference: http://doubango.org/API/tinyHTTP/

4.3.7 tinyXCAP

Prefix: `txcap`
API reference: http://doubango.org/API/tinyXCAP/

4.3.8 tinyMEDIA

Prefix: `tmedia`
API reference: http://doubango.org/API/tinyMEDIA/

4.3.9 tinySDP

Prefix: `tsdp`

API reference: http://doubango.org/API/tinySDP/

4.3.10 tinyMSRP

Prefix: `tmsrp`
API reference: http://doubango.org/API/tinyMSRP/

4.3.11 tinyGST

Prefix: `tgst`
API reference: http://doubango.org/API/tinyGST/

4.3.12 tinySIP

Prefix: `tsip`
API reference: http://doubango.org/API/tinySIP/

5 Debugging

The framework offers 4 debugging levels. The minimum level is defined by setting the value of DEBUG_LEVEL macro. Level values are:

- DEBUG_LEVEL_INFO: It's the minimum value. Used to log user information, application progress, configuration …
- DEBUG_LEVEL_WARN: Signal that something that could change the normal process happened. This type of error should not block the application.
- DEBUG_LEVEL_ERROR: It's the default minimum value. Signals that the current operation has failed. If the operation was critical (e.g. thread creation) then this will badly change the application behavior, otherwise (e.g. sending data over network) the application will continue to behave as expected.
- DEBUG_LEVEL_FATAL: It's the maximum value. This kind of errors is signaled just before the application crashes or enters in a state from which it's impossible to recover from.

All these macros will print the message to stderr. In your release program, you should define TSK_HAVE_DEBUG_H (#define TSK_HAVE_DEBUG_H 1) variable in order to supply your own debugging macros (or functions). Your macros (or functions) should be defined in a file named my_debug.h. This is typically used in conjunction with custom loggers (e.g. log4j, log4net, logfile…).

6 ANSI-C Object Programming

As you probably know, C is not an object oriented language.

Today, OOP (Object-Oriented Programing) is the best way to program well designed softwares.

In this document a “well-defined object” is a special C structure. All functions shown in this chapter are part of tinySAK project.

To explain how well-defined objects are implemented and used, I will give an example based on “Person” object.

The person object is declared like this:

```
typedef struct person_s
{
	TSK_DECLARE_OBJECT; /* Mandatory */

	char* name;
	struct person_s* girlfriend;
}
person_t;
```

6.1 Object Definition

An object definition could be considered as a class definition. The definition holds the object’s mandatory functions, size and a reference counter.

The mandatory functions are the `constructor`, the `destructor` and the `comparator`.

A C structure is #`define`d as an object by using `TSK_DECLARE_OBJECT` macro in its body.

A pointer to an object definition shall point to a `struct tsk_object_def_s`.

```
typedef struct tsk_object_def_s
{
	//! The size of the object.
	size_t size;
	//! Pointer to the constructor.
	tsk_object_t*  (* constructor) (tsk_object_t *, va_list *);
	//! Pointer to the destructor.
	tsk_object_t*  (* destructor) (tsk_object_t *);
	//! Pointer to the comparator.
	int             (*comparator) (const tsk_object_t *, const tsk_object_t *);
}
tsk_object_def_t;
```

An object is created in two phases. The first phase consists of dynamically allocating the object on the heap; this is why its size is mandatory in the object definition structure. When a new object is allocated on the heap, all its members (`char*`, `void*`, `int`, `long` …) will be zeroed. In the second phase, the newly created object will be initialized by calling the supplied `constructor`. To perform these two phases, you should call `tsk_object_new()` or `tsk_object_new_2()`.

The object is destroyed in two phases. The first phase consists of freeing its members (`void*`, `char*` …). It’s the `destructor` which is responsible of this task. In the second phase, the object itself is destroyed. As the object cannot destroy itself, you should use `tsk_object_unref` or `tsk_object_delete` to perform these two phases. The difference between these two functions is explained in the coming sections.

A well-defined object must never be freed using `free()` standard C function.

Below, an example of how to declare an object definition:

```
//(Object defnition)
 static const tsk_object_def_t person_def_t =
 {
	sizeof(person_t),
	person_ctor,
	person_dtor,
        person_cmp
  };
```

6.2 Constructor

The `constructor` is only responsible for the initialization and won't allocate the object. When passed to the constructor, the object is already allocated.

Here is an example:

```
// (constructor)
static tsk_object_t* person_ctor(tsk_object_t * self, va_list * app)
{
        person_t *person = self;
        if(person){
                person->name = tsk_strdup(va_arg(*app, const char *));
        }
        return self;
  }
```

6.3 Destructor

The `destructor` will free the object's members and won't destroy the object itself (Phase 1). The `destructor` function must return a pointer to itself to allow the caller to perform the second phase.

Here is an example:

```
// (destructor)
 static tsk_object_t * person_dtor(tsk_object_t * self)
 {
        person_t *person = self;
        if(person){
                TSK_FREE(person->name);
                tsk_object_unref(person->girlfriend);
        }
        return self;
  }
```

6.4 Comparator

The `comparator` function is used to compare two well-defined objects. The objects to compare shall have the same definition (or type).

Here is an example:

```
// (comparator)
static int person_cmp(const tsk_object_t *_p1, const tsk_object_t *_p2)
 {
        const person_t *p1 = _p1;
        const person_t *p1 = _p2;
        int ret;

        // do they have the same name?
        if((ret = tsk_stricmp(p1->name, p2->name))){
                return ret;
        }
        // do they have the same girlfriend?
        if((ret = tsk_object_cmp(p1->girlfriend, p2->girlfriend))){
                return ret;
        }

        // they are the same
        return 0;
  }
```

6.5 Reference counting

Reference counting is used to emulate garbage collection. Each well-defined object contains a reference counter field which indicates how many object have a reference to the actual object.

When an object is created (see below) the counter value is initialized to 1; this is automatically done and you have nothing to do. The counter is incremented by 1 when you call `tsk_object_ref` and decremented (by 1) when you call `tsk_object_unref`.

When the counter value reaches zero, then the object is garbaged (freed).

6.6 Inheritence

As you expect, inheritance is not supported in ANSI-C.

As any C Structure could be casted to a pointer to its first element, inheritance could be achieved like this:

```
#include "tsk.h"

// (a student is a person)
typedef struct student_s
{
        struct person_s* person; // Must be the first element
       char* school;
}
student_t;

// (as a student is a person you can do)
student_t* s;
((person_t*)s)->name = tsk_strdup("bob");
```

As `person_t` is a well-defined object, then `student_t` is also well-defined.

6.7 Usage

Once the object's definition is declared and all its mandatory functions implemented, it is used like this:

```
// creates a person: will call the constructor
person_t* bob = tsk_object_new(&person_def_t, "bob");
// creates bob's girlfriend
bob->girlfriend = tsk_object_new(&person_def_t, "alice");
// deletes bob: will delete both bob and bob's girlfriend field by calling their destructors
tsk_object_unref(bob);
```

As it's hard to guest which parameters the construct expects, it's common to use macro (or function) helpers. In our example the macro will look like this:

```
// create a person
#define PERSON_CREATE(name)  tsk_object_new(&person_def_t, (const char*)name)
```

As the destructor has fixed parameters, there is a common macro to destroy all kind of well-defined objects. `TSK_OBJECT_SAFE_FREE()` is used to destroy any object. The object will be freed only if; when decremented by 1 the reference count of the object is equal to zero. In all case (freed or not) the pointer value will be set to `NULL`.

The above example can be rewritten like this:

```
#include "tsk.h"

// create a person: will call the constructor
person_t* bob = PERSON_CREATE("bob");
// create bob's girlfriend
bob->girlfriend = PERSON_CREATE("alice");
// delete bob: will delete both bob and bob's girlfriend field by calling their destructors
TSK_OBJECT_SAFE_FREE(bob);
```

6.8 Lists

7 Threading

The framework provides an operating system agnostic threading functions for both WIN32 and Unix-like systems.

7.1 Threads

You don't need thousands of functions to manage threads. In the Framework we only need to create, pause and destroy threads.

Threads can be created using `tsk_thread_create()` and joined using `tsk_thread_join()`.

You can temporary cease the executing of a thread by calling `tsk_thread_sleep()`.

```
#include "tsk.h"

void* MyThreadFunction(void *arg)
{
        printf("arg=%d", *((int*)arg));
        return tsk_null;
}

void test_threads()
{
        void* tid[1] = {tsk_null}; // thread id
        int arg = 112; // arg to pass to the function

        // creates the thread
        tsk_thread_create(&tid[0], MyThreadFunction, &arg);

        // joins the thread
        tsk_thread_join(&(tid[0]));
}
```

7.2 Mutexes

Mutexes (Mutual exclusion) are used to protect a portion of code or function against concurrent access. Concurrent access happens when two or several threads try to execute the same portion of code at nearly the same time.

```
#include "tsk.h"

// create the mutext
tsk_mutex_handle_t *mutex = tsk_mutex_create();

tsk_mutex_lock(mutex);
// ...portion of code to protect
tsk_mutex_unlock(mutex);

// destroy the mutex
tsk_mutex_destroy(&mutex);
```

Mutexes are not well-defined objects; you should use `tsk_mutex_destroy()` instead of `TSK_OBJECT_SAFE_FREE()` to destroy them.

7.3 Thread-Safe Objects

Any C Structure could be declared as thread-safe using `TSK_DECLARE_SAFEOBJ` macro. It's not mandatory for the object to be well-defined.

A thread-safe object is initialized using `tsk_safeobj_init()` and deinitilized using `tsk_safeobj_deinit()`. To lock and unlock a portion of code which accesses the object you should use `tsk_safeobj_lock()` and `tsk_safeobj_unlock()` respectively.

7.4 Semaphores

Only counting semaphores are supported by the framework.

Counting semaphores are used to control the access to a portion of code which might be executed by multiple threads. A thread will have rights to execute the portion of code only if the semaphore's internal counter value is different than zero. Before executing the code to control, a thread should decrement the counter to check if it has permit.

```
#include "tsk.h"

// (create the semaphore)
tsk_semaphore_handle_t *sem = tsk_semaphore_create();
// (increment the counter)
tsk_semaphore_increment(sem);
// (decrement the counter)
tsk_semaphore_decrement(sem);
// (destoy the semaphore)
tsk_semaphore_destroy(&sem);
```

Semaphores are not well-defined objects; you should use `tsk_semaphore_destroy` instead of `TSK_OBJECT_SAFE_FREE()` to destroy them.

Mutexes are binary semaphores (counter value is always equals to 1 or 0).

7.5 Condition Variables

Condition variables are used to control the access to a portion of code which might be executed by multiple threads. Each thread will block until a certain condition is signaled or `ms` milliseconds have passed.

`tsk_condwait_create()` is used to create a condition variable, `tsk_condwait_wait()` to wait indefinitely until the condition is signaled and `tsk_condwait_timedwait(condition, ms)` to wait until the condition is signaled or `ms` milliseconds have passed.

`tsk_condwait_signal(condition)` is used to alert the first waiting thread that the condition is now true and `tsk_condwait_broadcast(condwait)` is used to alert all waiting threads.

Condition variables are not well-defined objects; you should use `tsk_condwait_destroy()` instead of `TSK_OBJECT_SAFE_FREE()` to destroy them.

7.6 Runnable

A `runnable` object is a well-defined object and is declared using `TSK_DECLARE_RUNNABLE` macro.

A `runnable` object must be explicitly started using `tsk_runnable_start()` and is implicitly stopped when destroyed. You can explicitly stop the object by calling `tsk_runnable_stop()`.

8 Final Sate Machine

9 Timer Manager

10 Sockets and Network Functions

All network functions are part of tinyNET projects.

You MUST call `tnet_startup()` before using any network function (`tnet_*`). `tnet_cleanup()` is used to terminate use of network functions. The startup function will determine whether the host is a “little-endian” machine or not (at runtime). To have access to all networking functions, you should #include `“tinynet.h”` file in your project.

10.1 Sockets

For performance reason, all sockets created using tinyNET are non-blocking by default. The newly created socket will be automatically bound to associate it with an IP address and port number. `tnet_socket_create()` macro is used to create and bind a non-blocking socket. Use `tnet_socket_create_2()` macro to control whether the socket should be bound or not. The same macro is used to force the stack to create a blocking socket.

A socket object is defined like this:

```
typedef struct tnet_socket_s
{
	TSK_DECLARE_OBJECT;

	tnet_socket_type_t type;
	tnet_fd_t fd;
	tnet_ip_t ip;
	uint16_t port;

	tnet_tls_socket_handle_t* tlshandle;
}
tnet_socket_t;
```

To create a socket:

```
// (create udp ipv4 or ipv6 socket)
tnet_socket_t* socket = tnet_socket_create(
		TNET_SOCKET_HOST_ANY, // local ip address/hostname to bind to
		TNET_SOCKET_PORT_ANY, // local port number to bind to
		tnet_socket_type_udp_ipv46 // the socket type (IPv4 or IPv6)
		);

// TNET_SOCKET_HOST_ANY --> bind to "0.0.0.0" or "::"
// TNET_SOCKET_PORT_ANY --> bind to any available port
```

`TNET_SOCKET_TYPE_IS_*` macros are used to determine:

- The socket type (stream, dgram),
- The socket protocol (udp, tcp, tls, sctp, ipsec),
- The IP version (ipv6, ipv4),
- …

A socket is a well-defined object and should be destroyed using `TSK_DECLARE_SAFE_FREE()` macro.

A socket will be automatically closed when destroyed.

10.2 Tansport

A transport layer always has a master socket which determine what kind of network traffic we expect (stream or dgram). Stream transport can manage TCP, TLS and SCTP sockets. Datagram socket can only manage UDP sockets. A transport can hold both IPv4 and IPv6 sockets.

11 DNS

The DNS Stack (RFC 1034 and RFC 1035) contains all network functions to send queries (both

IPv4 and IPv6) and parse responses. The core framework implements RFC 3401, 3402, 3403 and 3404, also known as Dynamic Delegation Discovery System (DDDS).

The DNS servers are automatically loaded by the stack when you create a context (`tnet_dns_ctx_create()`). On Windows systems (XP, VISTA, 7 and CE) the servers are retrieved using WIN32 APIs. On Unix-like systems (both desktop and embedded) such as Debian, Ubuntu, Nokia's N900… the list of DNS servers comes from "`/etc/resolv.conf`" file. On Google Android operating system, this file is missing and the DNS settings are stored in the shared memory. You can access this shared memory by using `property_get()` and `property_set()` function which are part of Bionic. In all cases, you can retrieve the DNS servers yourself (e.g. using java/C# Frameworks) and add them to the context using `tnet_dns_add_server(ctx, host)`.

DNS resolution is always performed in a synchronous manner and is thread-safe. For all DNS requests the default timeout value is 5 seconds (`TNET_DNS_TIMEOUT_DEFAULT`).

The stack also implements the ENUM protocol (RFC 3761).

11.1 Resource Records

The table below lists all DNS Resource Records (RR) for which we provide a parser.

Code	RFC	Description	Well-defined type
A	RFC 1035	IPv4 address	`tnet_dns_a_t`
AAAA	RFC 3596	IPv6 address	`tnet_dns_aaaa_t`
CNAME	RFC 1035	Canonical name	`tnet_dns_cname_t`
MX	RFC 2035	Mail exchange	`tnet_dns_mx_t`
NAPTR	RFC 3403	Naming Authority Pointer	`tnet_dns_naptr_t`
NS	RFC 1035	Name Server	`tnet_dns_ns_t`
OPT	RFC 2671	Option	`tnet_dns_opt_t`
PTR	RFC 1035	Pointer record	`tnet_dns_ptr_t`
SOA	RFC1035	Start Of Authority record	`tnet_dns_soa_t`
SRV	RFC 2782	Service locator	`tnet_dns_srv_t`

Here is an example of how to use the DNS stack to perform DNS NAPTR resolution and print the result to the console.

```
tnet_dns_ctx_t *ctx = tnet_dns_ctx_create();
tnet_dns_response_t *response = tsk_null;
const tsk_list_item_t* item;
const tnet_dns_rr_t* rr;

if((response = tnet_dns_resolve(ctx, "sip2sip.info", qclass_in, qtype_naptr)))
{
        if(TNET_DNS_RESPONSE_IS_SUCCESS(response)){
                TSK_DEBUG_INFO("We got a success response from the DNS server.");
                // loop through the answers
                tsk_list_foreach(item, response->Answers){
                        rr = item->data;
                        if(rr->qtype == qtype_naptr){
                                const tnet_dns_naptr_t *naptr = (const tnet_dns_naptr_t*)rr;

                                TSK_DEBUG_INFO("order=%u pref=%u flags=%s services=%s regexp=%s re-
placement=%s",
                                        naptr->order,
                                        naptr->preference,
                                        naptr->flags,
                                        naptr->services,
                                        naptr->regexp,
                                        naptr->replacement);
                        }
                }
        }
        else{
                TSK_DEBUG_ERROR("We got an error response from the DNS server. Error code: %u", re-
sponse->Header.RCODE);
        }
}

TSK_OBJECT_SAFE_FREE(response);
TSK_OBJECT_SAFE_FREE(ctx);
```

The `ctx` could be used several times and is a well-defined object.

The console will output:

```
*INFO: We got a success response from the DNS server.
*INFO: order=10 pref=0 flags=S services=SIP+d2u regexp=(null) replacement=_sip._udp.sip2sip.info
```

You should use tnet_dns_query_naptr_srv() to send a *DNS NAPTR* request followed by a *DNS SRV* request. This function is useful for SIP/IMS clients trying to quickly discover the Proxy-CSCF address without having to deal with many functions.

11.2 ENUM

ENUM (E.164 Number Mapping) protocol has been defined in RFC 3761.

ENUM protocol is used to transform telephone numbers of the PSTN network (e.g. +33600000) into internet resource addresses (e.g. sip:diopmamadou@example.com) by using DNS lookup (NAPTR). The internet resource address could be an email, ICQ, IAX2, H.323 …

In our case (3GPP IMS) it is typically used to convert TEL URIs (e.g. tel:+33600000) into SIP URIs (sip:+33600000;user=phone). The telephone number to convert should be a valid E.164 number.

Useful website for testing: http://enumquery.com

11.2.1 Usage

The code below shows how to gets the SIP address (with the higher order) associated to an E.164 telephone number.

```
tnet_dns_ctx_t *ctx = tnet_dns_ctx_create();
tnet_dns_response_t* response = tsk_null;
char* uri = tsk_null;

if((uri = tnet_dns_enum_2(ctx, "E2U+SIP", "+1-800-555-5555","e164.org"))){
        TSK_DEBUG_INFO("URI=%s", uri);
        TSK_FREE(uri);
}
else{
        TSK_DEBUG_ERROR("ENUM(%s) failed", "+1-800-555-5555");
}

TSK_OBJECT_SAFE_FREE(response);
TSK_OBJECT_SAFE_FREE(ctx);
```

Console Output:

```
*INFO: URI=sip:16416418000-555-5555@sip.tollfreegateway.com
```

`E2U+SIP` is the name of SIP ENUM service assigned by the IANA. Any assigned service () could be used even if the associated addresse type isn't a well-knonw internet address.

To get all internet addresses (email, IAX2, ICQ, H.323 …) associated to the telephone, use `tnet_dns_enum()` instead of `tnet_dns_enum_2()`.

12 DHCPv4

13 DHCPv6

14 NAT Traversal

14.1 STUN2

14.2 TURN

14.3 ICE

15 HTTP/HTTPS

The HTTP/HTTPS stack is a basic thread-safe client API and is used in conjunction with the XCAP protocol. Almost all HTTP methods such as OPTIONS, HEAD, GET, DELETE, POST, CONNET, TRACE or PUT … are supported for outgoing requests. Only response messages will be correctly handled by the stack. Requests will be passed to the application layer "as is" and no new connection will be opened.

Both IPv4 and IPv6 are supported. If the host name (FQDN) resolution leads to both IPv4 and IPv6 results, then IPv4 will be used by default.

When HTTPS is used, then both end-user and mutual authentication modes are supported. For mutual authentication the TLS/SSL certificate files MUST be sets by using `THTTP_STACK_SET_TLS_CERTS(CA_FILE_STR, PUB_FILE_STR, PRIV_FILE_STR).`

15.1 Initialization

As the HTTP/HTTPS stack depends on the network library (tinyNET), you MUST call `tnet_startup()` before using any HTTP/Network function (`thttp_*`). `tnet_cleanup()` is used to terminate use of network functions. To have access to all XCAP functions, you should #include `"tinyxcap.h"` file in your project.

The example below demonstrates how to create and start a HTTP/HTTPS stack. The callback function shows the most common events.

```
int ret;

int test_stack_callback(const thttp_event_t *httpevent)
{
       switch(httpevent->type){
              case thttp_event_message: /* New HTTP message */
                     {
                            break;
                     }
              case thttp_event_auth_failed: /* Authentication failed */
                     {
                            break;
                     }


              case thttp_event_closed: /* HTTP connection closed (informational) */
                     {
                            break;
                     }
       }
```

```
        return 0;
}

thttp_stack_handle_t* stack = thttp_stack_create(test_stack_callback,
        // TLS certificates (not mandatory) to show how parameters are passed to the stack
        THTTP_STACK_SET_TLS_CERTS("C:\\tls\\ca.pki-crt.pem", "C:\\tls\\pub-crt.pem", "C:\\tls\\pub-
key.pem"),
        THTTP_STACK_SET_NULL());/* MUST always be present */

if((ret = thttp_stack_start(stack))){
        TSK_DEBUG_ERROR("Failed to start the HTTP/HTTPS stack.");
        goto bail;
}
```

A stack is a well-defined object and must be destroyed by using `TSK_OBJECT_SAFE_FREE()` macro.

15.2 Sessions

A session could be seen as a peer2peer persistent connection and will be maintained by the stack as long as you wish to keep the network connection opened (not explicitly destroyed). If the connection is closed by the remote peer, then the stack will automatically reopen it when you try to send a new HTTP/HTTP request. The network connection will be definitely closed when the session is destroyed.

As the connection is persistent, then you can send multiple requests without waiting for each response. This mode is called “Pipelining” and is defined as per RFC 2616 section 8.1.2.2.

You should not pipeline requests using non-idempotent methods or non-idempotent sequences of methods. This means that you can safely pipeline GET or HEAD methods but should not with PUT or POST requests. Only HTTP version 1.1(or later) requests should be pipelined.

To avoid pipelining, you must use a session object (`thttp_session_handle_t*`) only once to send a single request.

The example below shows how to create and configure a session.

```
/* creates session */
thttp_session_handle_t * session = thttp_session_create(stack,
        // session-level credentials
         THTTP_SESSION_SET_CRED("ali baba", "open sesame"),

        // session-level options
        THTTP_SESSION_SET_OPTION(THTTP_SESSION_OPTION_TIMEOUT, "6000"),

        // session-level headers
        THTTP_SESSION_SET_HEADER("Pragma", "No-Cache"),
        THTTP_SESSION_SET_HEADER("Connection", "Keep-Alive"),
        THTTP_SESSION_SET_HEADER("User-Agent", "doubango 1.0"),

        THTTP_SESSION_SET_NULL());/* MUST always be present */
```

A session is a well-defined object and must be destroyed by using `TSK_OBJECT_SAFE_FREE()` macro.

15.2.1 Credentials

Both HTTP digest and Basic authentication (RFC 2617) are supported in this version (1.0). The credentials (username and password) should be set when the session is created (`thttp_session_create()`) or later, by using `thttp_session_set()`. As credentials are configured at session level, you can use one stack to authenticate against multiple HTTP servers with different domains.

The stack will automatically add authorizations (as per RFC 2617) when it is challenged (401/407 responses), this is why you should set credentials before sending any requests which is susceptible to be challenged.

15.2.2 Options

All options shall be set by using `THTTP_SESSION_SET_OPTION(id, value)` macro. Session-level options will be applied to all underlying requests, unless the request redefines this option.

For more information, please visit the website.

15.2.3 Headers

All headers shall be set by using `THTTP_SESSION_SET_HEADER(name, value)` macro. Session-level headers will be added to all underlying requests even if a request redefines this header. This means that if both the request and the session have the same header, then it will be duplicated.
Host and *Content-Length* headers are automatically added by the stack.
`THTTP_SESSION_UNSET_HEADER()` and `THTTP_SESSION_SET_HEADER`() macros are used to remove or update a previously added session-level header, respectively.

For more information, please visit the website.

15.3 Requests

A HTTP request is referred to as an "**action**" and is always associated to a session. There are nine well-know actions you can perform: `CONNET`, `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST`, `PUT` and `TRACE`. You can use `thttp_action_perform(session, url, method, …)` function to send a custom request. All `thttp_action_*()` functions are non-blocking.

All requests are sent in an asynchronous manner and the result (HTTP messages, errors, time out …) will be passed to the callback function.

The code below shows how to send `HTTP PUT` request.

```
int ret = thttp_action_PUT(session, "http://www.doubango.org",
              // action-level options
              THTTP_ACTION_SET_OPTION(THTTP_ACTION_OPTION_TIMEOUT, "2500"),

              // payload
              THTTP_ACTION_SET_PAYLOAD("Comment allez-vous?", strlen("Comment allez-vous?")),

              // action-level headers
              THTTP_ACTION_SET_HEADER("Content-Type", "text/plain"),
              THTTP_ACTION_SET_HEADER("Expect", "100-continue"),

              THTTP_ACTION_SET_NULL());
```

The code below uses `HTTP GET` request to connect to an IPv6 website

```
int ret = thttp_action_GET(session, "http://ipv6.google.com",
              // action-level options
              THTTP_ACTION_SET_OPTION(THTTP_ACTION_OPTION_TIMEOUT, "4500"),

              THTTP_ACTION_SET_NULL());
```

You can notice that, there is nothing special to do in order to connect to an IPv6 website.

15.3.1 Options

All options shall be set by using `THTTP_ACTION_SET_OPTION(id, value)` macro. Action-level options and headers are only applied to the current request. In the code, the timeout previously defined by the session has been redefined (from 6000ms to 2500ms).

For more information, please visit the website.

15.3.2 Headers

All headers shall be set by using `THTTP_ACTION_SET_HEADER(name, value)` macro. Action-level headers will only be added to the current request. *Host* and *Content-Length* headers are automatically added by the stack.
For more information, please visit the website.

16 XCAP

The XCAP Framework is mainly based on RFC 4825 and uses tinyHTTP project. The framework can be used to implements advanced OMA functionalities such Enhanced Address Book, Presence Authorization Rules, Service Configuration …

At startup the stack will load all supported AUIDs with their default values.

16.1 Initialization

As the XCAP stack depends on the HTTP/HTTPS stack (tinyHTTP) which uses the network library (tinyNET), you MUST call `tnet_startup()` before using any XCAP function (`txcap_*`). `tnet_cleanup()` is used to terminate use of network functions.

The example below demonstrates how to create and start a XCAP stack. In this example, `the xcap-root` URI is http://doubango.org:8080/services and the SIP AOR (used as XUI) is `sip:bob@doubango.org`.

```
txcap_stack_handle_t* stack = tsk_null;
int ret;

tnet_startup();

stack = txcap_stack_create(test_stack_callback, "sip:bob@doubango.org", "mysecret", "http://dou-
bango.org:8080/services",

        // options
        TXCAP_STACK_SET_OPTION(TXCAP_STACK_OPTION_TIMEOUT, "6000"),

        // stack-level headers (not mandatory)
        TXCAP_STACK_SET_HEADER("Pragma", "No-Cache"),
        TXCAP_STACK_SET_HEADER("Connection", "Keep-Alive"),
         TXCAP_STACK_SET_HEADER("X-3GPP-Intended-Identity", "sip:bob@doubango.org"),
        TXCAP_STACK_SET_HEADER("User-Agent", "XDM-client/OMA1.1"),
        TXCAP_STACK_SET_NULL());

if((ret = txcap_stack_start(stack))){
        goto bail;
}

// …………

bail:
        TSK_OBJECT_SAFE_FREE(stack);

tnet_cleanup();
```

The stack-level headers will be added in all outgoing requests.

A stack is a well-defined object and must be destroyed by using `TSK_OBJECT_SAFE_FREE()` macro. The stack will be automatically stopped when destroyed.

16.2 Application Unique ID (AUID) object

An AUID object is defined by:

- An id (e.g. “xcap-caps”),
- A MIME-Type (e.g. “application/xcap-caps+xml”),
- A namespace (e.g. “urn:ietf:params:xml:ns:xcap-caps”),
- A document name (e.g. “index”), which defines the name of the default document associated with this AUID
- A scope (“global” or “users”)

At startup, the stack will load all supported AUIDs with their default values. You can at any time change these values or register your own AUID object. The list of default AUIDs with their default values are shown in the next sections.

When you are about to send a request, it's not mandatory to use a registered AUID but it's easier to generate the selector as all parameters are pre-configured.

16.2.1 Default AUIDs

The table below shows the default AUIDs as they are defined by the stack at startup.

Id	MIME-Type	Namespace	Document	Scope	Reference
xcap-caps	application/xcap-caps+xml	urn:ietf:params:xml:ns:xcap-caps	index	global	RFC 4825 section 12.1
resource-lists	application/resource-lists+xml	urn:ietf:params:xml:ns:resource-lists	index	users	RFC 4826 section 3.4.1
rls-services	application/rls-services+xml	urn:ietf:params:xml:ns:rls-services"	index	users	RFC 4826 section 4.4.1
pres-rules	application/auth-policy+xml	urn:ietf:params:xml:ns:pres-rules	index	users	RFC 5025 section 9.1
org.openmobilealliance.pres-rules	application/auth-policy+xml	urn:ietf:params:xml:ns:common-policy	pres-rules	users	[OMA-TS-Presence_SIMPLE_XDM-V1_1-20080627-A] section 5.1.1.2
directory	application/directory+xml	urn:ietf:params:xml:ns:xcap-directory	directory.xml	users	draft-garcia-simple-xcap-directory-00 section 9.1
org.openmobilealliance.xcap-directory	application/vnd.oma.xcap-directory+xml	urn:oma:xml:xdm:xcap-directory	directory.xml	users	[OMA-TS-XDM_Core-V1_1-20080627-A] section 6.7.2.1
org.openmobilealliance.pres-content	application/vnd.oma.pres-content+xml	urn:oma:xml:prs:pres-content	oma_status-icon/rcs_status_icon	users	[OMA-TS-Presence-SIMPLE_Content_XDM-V1_0-20081223-C] section 5.1.2
org.openmobilealliance.conv-history	application/vnd.oma.im.history-list+xml	urn:oma:xml:im:history-list	conv-history	users	[OMA-TS-IM_XDM-V1_0-20070816-C] section 5.1.2
org.openmobilealliance.deferred-list	application/vnd.oma.im.deferred-list+xml	urn:oma:xml:im:history-list	deferred-list	users	[OMA-TS-IM_XDM-V1_0-20070816-C] section 5.2.2
org.openmobilealliance.group-usage-list	application/vnd.oma.group-usage-list+xml	rn:ietf:params:xml:ns:resource-lists	index	users	[OMA-TS-XDM_Shared-V1_1-20080627-A] subclause 5.2.2

16.2.2 AUID registration

The code below shows how to register two AUIDs. If the AUID object already exist (case-insensitive comparison on the id), then it will be updated. All fields are mandatory (id, mime-type, namespace, document and scope).

```
txcap_stack_set(stack,
       TXCAP_STACK_SET_AUID("my-xcap-caps", "application/my-xcap-caps+xml", 
"urn:ietf:params:xml:ns:my-xcap-caps", "my-document", tsk_true),
       TXCAP_STACK_SET_AUID("my-resource-lists", "application/my-resource-lists+xml", 
"urn:ietf:params:xml:ns:my-resource-lists", "my-document", tsk_false),

       TXCAP_STACK_SET_NULL());/* mandatory */
```

The stack should be created as shown at section 16.1.

Only AUIDs which don't appear in the table above should be registered using this method

16.3 Selector

The selector is a helper function which could be used to construct XCAP URIs. XCAP URI is constructed as per RFC 4825 section 6. `TXCAP_SELECTOR_NODE_SET*()` macros are used to build a complete and well-formed URI (already percent encoded).

All examples below assume that our `AOR (used as XUI)` is `sip:bob@doubango.com`, we are using the `'rcs'` list and the `xcap-root` URI is http://doubango.org:8080/services. All these parameters should be set when the stack is created. You will also notice that `TXCAP_SELECTOR_NODE_SET_NULL()` macro is used to ends the node selection parameters passed to `txcap_selector_get_url()`, it's mandatory and should always be the last one.

- Select XDMS capabilities:

```
char* urlstring = txcap_selector_get_url(stack, "xcap-caps",
              TXCAP_SELECTOR_NODE_SET_NULL());
       TSK_DEBUG_INFO("%s\n", urlstring);
       TSK_FREE(urlstring);
```

Console Output:

```
http://doubango.org:8080/services/xcap-caps/global/index
```

- Select 'resource-lists' document

```
char* urlstring = txcap_selector_get_url(stack, "resource-lists",
              TXCAP_SELECTOR_NODE_SET_NULL());
TSK_DEBUG_INFO("%s\n", urlstring);
TSK_FREE(urlstring);
```

Console Output:

```
http://doubango.org:8080/services/resource-lists/users/sip:bob@doubango.org/index
```

- Select 'rcs' list

```
char* urlstring = txcap_selector_get_url(stack, "resource-lists",
              TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("list", "name", "rcs"),
              TXCAP_SELECTOR_NODE_SET_NULL());
TSK_DEBUG_INFO("%s\n", urlstring);
```

Console Output:

```
http://doubango.org:8080/services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-lists/list%5B@name=%22rcs%22%5D
```

- Select the 2nd list

```
char* urlstring = txcap_selector_get_url(stack, "resource-lists",
              TXCAP_SELECTOR_NODE_SET_POS("list", 2),
              TXCAP_SELECTOR_NODE_SET_NULL());
TSK_DEBUG_INFO("%s\n", urlstring);
TSK_FREE(urlstring);
```

Console Output:

```
http://doubango.org:8080/services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-lists/list%5B2%5D
```

- Select the 4th list using wildcard

```
urlstring = txcap_selector_get_url(stack, "resource-lists",
              TXCAP_SELECTOR_NODE_SET_POS("*", 4),
              TXCAP_SELECTOR_NODE_SET_NULL());
TSK_DEBUG_INFO("%s\n", urlstring);
TSK_FREE(urlstring);
```

Console Output:

```
http://doubango.org:8080/services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-lists/*%5B4%5D
```

- Select bob's entry

```
char* urlstring = txcap_selector_get_url(stack, "resource-lists",
              TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("list", "name", "rcs"),
              TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("entry", "uri", "sip:bob@doubango.com"),
              TXCAP_SELECTOR_NODE_SET_NULL());
TSK_DEBUG_INFO("%s\n", urlstring);
TSK_FREE(urlstring);
```

Console Output:

```
http://doubango.org:8080/services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-
lists/list%5B@name=%22rcs%22%5D/entry%5B@uri=%22sip:bob@doubango.org%22%5D
```

- Select bob's display-name

```
char* urlstring = txcap_selector_get_url(stack, "resource-lists",
              TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("list", "name", "rcs"),
              TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("entry", "uri", "sip:bob@doubango.org"),
              TXCAP_SELECTOR_NODE_SET_NAME("display-name"),
              TXCAP_SELECTOR_NODE_SET_NULL());
TSK_DEBUG_INFO("%s\n", urlstring);
TSK_FREE(urlstring);
```

Console Output:

```
http://doubango.org:8080/services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-
lists/list%5B@name=%22rcs%22%5D/entry%5B@uri=%22sip:bob@doubango.org%22%5D/display-name
```

- Select the display-name of the 1st entry

```
char* urlstring = txcap_selector_get_url(stack, "resource-lists",
              TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("list", "name", "rcs"),
              TXCAP_SELECTOR_NODE_SET_POS("entry", 1),
              TXCAP_SELECTOR_NODE_SET_NAME("display-name"),
              TXCAP_SELECTOR_NODE_SET_NULL());
TSK_DEBUG_INFO("%s\n", urlstring);
TSK_FREE(urlstring);
```

Console Output:

```
http://doubango.org:8080/services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-
lists/list%5B@name=%22rcs%22%5D/entry%5B1%5D/display-name
```

- Select bob from position 23

```
char* urlstring = txcap_selector_get_url(stack, "resource-lists",
              TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("list", "name", "rcs"),
              TXCAP_SELECTOR_NODE_SET_POS_ATTRIBUTE("entry", 23, "uri", "sip:bob@doubango.org"),
              TXCAP_SELECTOR_NODE_SET_NULL());
TSK_DEBUG_INFO("%s\n", urlstring);
TSK_FREE(urlstring);
```

Console Output:

```
http://doubango.org:8080/services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-
lists/list%5B@name=%22rcs%22%5D/entry%5B23%5D%5B@uri=%22sip:bob@doubango.org%22%5D
```

- Namespaces test

```
char* urlstring = txcap_selector_get_url(stack, "resource-lists",
              TXCAP_SELECTOR_NODE_SET_NAME("foo"),
              TXCAP_SELECTOR_NODE_SET_NAME("a:bar"),
              TXCAP_SELECTOR_NODE_SET_NAME("b:baz"),
              TXCAP_SELECTOR_NODE_SET_NAMESPACE("a", "urn:namespace1-uri"),
              TXCAP_SELECTOR_NODE_SET_NAMESPACE("b", "urn:namespace2-uri"),
              TXCAP_SELECTOR_NODE_SET_NULL());
TSK_DEBUG_INFO("%s\n", urlstring);
TSK_FREE(urlstring);
```

Console Output:

```
http://doubango.org:8080/services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-
lists/foo/a:bar/b:baz%3Fxmlns(a=%22urn:namespace1-uri%22)xmlns(b=%22urn:namespace2-uri%22)
```

16.4 XDMC Usage

It is assumed that the address of the XDMS (or aggregation Proxy) is "`doubango.org:8080/services`" and thus the XCAP Root URI is "`doubango.org:8080/services`". "`sip:bob@doubango.org`" will be used as the XUI.

An XDMC can perform twelve actions:

- `txcap_action_create_element`: Creates new element by sending a `HTTP/HTTPS PUT` request. The default *Content-Type* will be "`application/xcap-el+xml`", unless you provide your own *Content-Type* by using `TXCAP_ACTION_SET_HEADER()`.

- `txcap_action_create_document`: Creates new document by sending a `HTTP/HTTPS PUT` request. The default *Content-Type* will be the one associated with the AUID of the document, unless you provide your own *Content-Type* by using `TXCAP_ACTION_SET_HEADER()`.

- `txcap_action_create_attribute`: Creates new attribute by sending a `HTTP/HTTPS PUT` request. The default *Content-Type* will be "`application/xcap-att+xml`", unless you provide your own *Content-Type* by using `TXCAP_ACTION_SET_HEADER()`.

- `txcap_action_replace_element`: Replaces an element by sending a `HTTP/HTTPS PUT` request. The default *Content-Type* will be "`application/xcap-el+xml`", unless you provide your own *Content-Type* by using `TXCAP_ACTION_SET_HEADER()`.

- `txcap_action_replace_document`: Replaces a document by sending a `HTTP/HTTPS PUT` request. The default *Content-Type* will be the one associated with the AUID of the document, unless you provide your own *Content-Type* by using `TXCAP_ACTION_SET_HEADER()`.

- `txcap_action_replace_attribute`: Replaces an attribute by sending a `HTTP/HTTPS PUT` request. The default *Content-Type* will be "`application/xcap-att+xml`", unless you provide your own *Content-Type* by using `TXCAP_ACTION_SET_HEADER()`.

- `txcap_action_fetch_element`: Retrieves an element from the XDMS by sending a `HTTP/HTTPS GET` request. The default *Content-Type* will be "`application/xcap-el+xml`", unless you provide your own *Content-Type* by using `TXCAP_ACTION_SET_HEADER()`.

- `txcap_action_fetch_document`: Retrieves a document from the XDMS by sending a `HTTP/HTTPS GET` request. The default *Content-Type* will be the one associated with the AUID of the document, unless you provide your own *Content-Type* by using `TXCAP_ACTION_SET_HEADER()`.

- `txcap_action_fetch_attribute`: Retrieves an attribute from the XDMS by sending a `HTTP/HTTPS GET` request. The default *Content-Type* will be "`application/xcap-att+xml`", unless you provide your own *Content-Type* by using `TXCAP_ACTION_SET_HEADER()`.

- `txcap_action_delete_element`: Deletes an element from the XDMS by sending a `HTTP/HTTPS DELETE` request.

- `txcap_action_delete_document`: Deletes a document from the XDMS by sending a `HTTP/HTTPS DELETE` request.

- `txcap_action_delete_attribute`: Deletes an attribute from the XDMS by sending a `HTTP/HTTPS DELETE` request.

To understand how the stack is created, please refer to section 16.1.

16.4.1 Retrieving XDMS capabilities

The code below shows how an XDMC obtains the XDMS capabilities document.

```
int ret = txcap_action_fetch_document(stack,
              // selector
              TXCAP_ACTION_SET_SELECTOR("xcap-caps",
                     TXCAP_SELECTOR_NODE_SET_NULL()),
              // ends parameters
              TXCAP_ACTION_SET_NULL()
              );
```

The XDMC will send:

```
GET /services/xcap-caps/global/index HTTP/1.1
Host: doubango.org:8080
Connection: Keep-Alive
User-Agent: XDM-client/OMA1.1
X-3GPP-Intended-Identity: sip:bob@doubango.org
Content-Type: application/xcap-caps+xml
```

16.4.2 Address Book

- The code below shows how an XDMC obtains URI Lists (Address Book).

```
int ret = txcap_action_fetch_document(stack,
              // action-level options
              TXCAP_ACTION_SET_OPTION(TXCAP_ACTION_OPTION_TIMEOUT, "6000"),
              //action-level headers
              TXCAP_ACTION_SET_HEADER("Pragma", "No-Cache"),
              // selector
              TXCAP_ACTION_SET_SELECTOR("resource-lists",
                     TXCAP_SELECTOR_NODE_SET_NULL()),
              // ends parameters
              TXCAP_ACTION_SET_NULL()
              );
```

The XDMC will send:

```
GET /services/resource-lists/users/sip:bob@doubango.org/index HTTP/1.1
Host: doubango.org:8080
Connection: Keep-Alive
User-Agent: XDM-client/OMA1.1
X-3GPP-Intended-Identity: sip:bob@doubango.org
Pragma: No-Cache
Content-Type: application/resource-lists+xml
```

- The code below shows how to add a new list to the address book

```
int ret = txcap_action_create_element(stack,
              // selector
              TXCAP_ACTION_SET_SELECTOR("resource-lists",
                     TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("list", "name", "newlist"),
                     TXCAP_SELECTOR_NODE_SET_NULL()),
              // payload
              TXCAP_ACTION_SET_PAYLOAD(PAYLOAD, strlen(PAYLOAD)),
              // ends parameters
              TXCAP_ACTION_SET_NULL()
              );
```

The XDMC will send:

```
PUT /services/resource-lists/users/sip:bob@doubano.org/index/~~/resource-lists/list%5B@name=%22newl-
ist%22%5D HTTP/1.1
Host: doubango.org:8080
Content-Length: 110
Connection: Keep-Alive
User-Agent: XDM-client/OMA1.1
X-3GPP-Intended-Identity: sip:bob@doubango.org
Content-Type: application/xcap-el+xml

<list name="newlist" xmlns="urn:ietf:params:xml:ns:resource-lists"><display-name>newlist</display-
name></list>
```

- The code below shows how to retrieve the previously added list

```
int ret = txcap_action_fetch_element(stack,
              // action-level selector
              TXCAP_ACTION_SET_SELECTOR("resource-lists",
                     TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("list", "name", "newlist"),
                     TXCAP_SELECTOR_NODE_SET_NULL()),
              // ends parameters
              TXCAP_ACTION_SET_NULL()
              );
```

The XDMC will send:

```
GET /services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-lists/list%5B@name=
%22newlist%22%5D HTTP/1.1
Host: doubango.org:8080
Connection: Keep-Alive
User-Agent: XDM-client/OMA1.1
X-3GPP-Intended-Identity: sip:bob@doubango.org
Content-Type: application/xcap-el+xml
```

- The code below shows how to add a new entry (“`sip:alice@doubango.org`”) to the previously added list

```
int ret = txcap_action_create_element(stack,
              // selector
```

```
               TXCAP_ACTION_SET_SELECTOR("resource-lists",
                       TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("list", "name", "newlist"),
                       TXCAP_SELECTOR_NODE_SET_ATTRIBUTE("entry", "uri", “sip:alice@doubango.org”),
                       TXCAP_SELECTOR_NODE_SET_NULL()),
               // payload
               TXCAP_ACTION_SET_PAYLOAD(PAYLOAD, strlen(PAYLOAD)),
               // ends parameters
               TXCAP_ACTION_SET_NULL()
               );
```

The XDMC will send:

```
PUT /services/resource-lists/users/sip:bob@doubango.org/index/~~/resource-lists/list%5B@name=
%22newlist%22%5D/entry%5B@uri=%22sip:alice@doubango.org%22%5D HTTP/1.1
Host: doubango.org:8080
Content-Length: 125
Connection: Keep-Alive
User-Agent: XDM-client/OMA1.1
X-3GPP-Intended-Identity: sip:bob@doubango.org
Content-Type: application/xcap-el+xml

<entry uri="sip:alice@doubango.org" xmlns="urn:ietf:params:xml:ns:resource-lists"><display-
name>alice</display-name></entry>
```

16.4.3 Obtaining Presence Content Document

The code below shows how an XDMC obtains the Presence Content document (avatar).

```
int ret = txcap_action_fetch_document(stack,
               // action-level options
               TXCAP_ACTION_SET_OPTION(TXCAP_ACTION_OPTION_TIMEOUT, "6000"),
               //action-level headers
               TXCAP_ACTION_SET_HEADER("Pragma", "No-Cache"),
               // selector
               TXCAP_ACTION_SET_SELECTOR("org.openmobilealliance.pres-content",
                       TXCAP_SELECTOR_NODE_SET_NULL()),
               // ends parameters
               TXCAP_ACTION_SET_NULL()
               );
```

The XDMC will send:

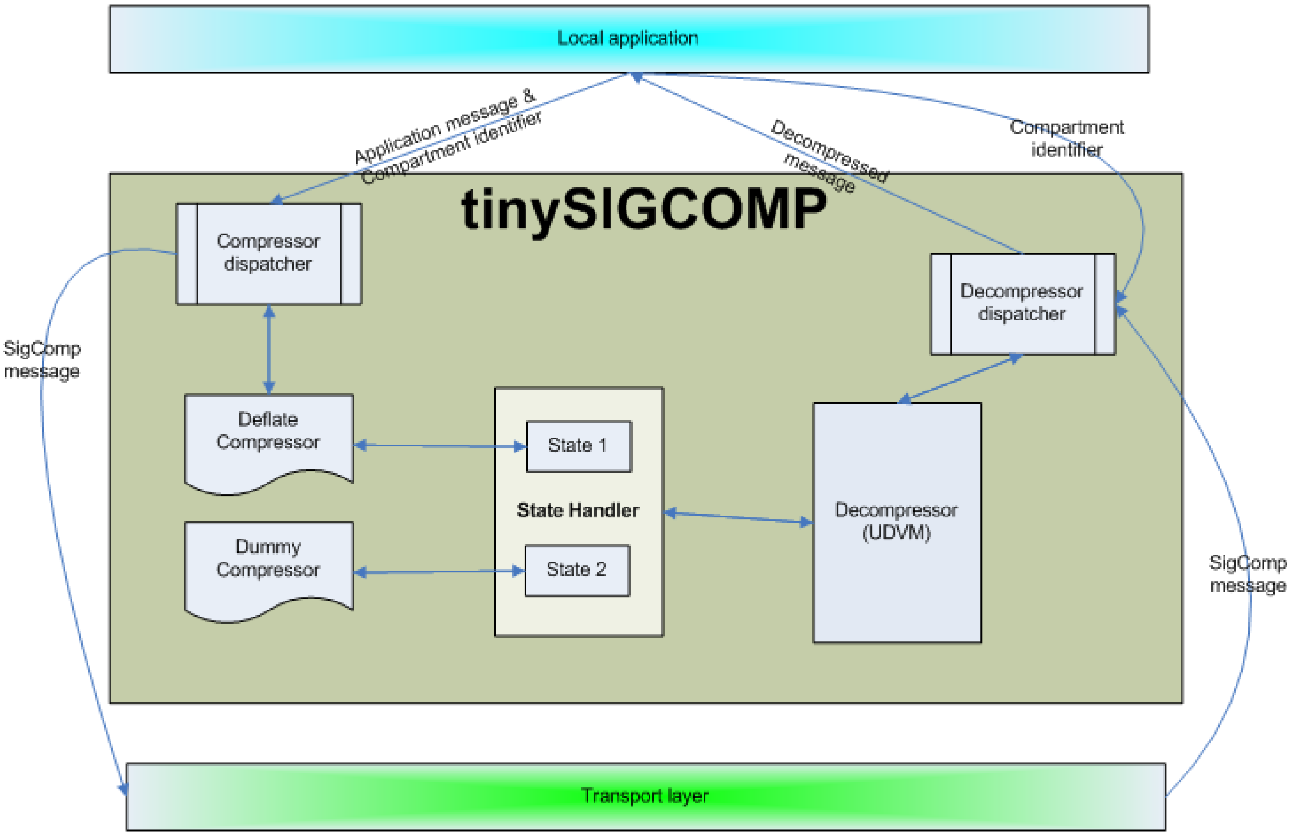
```
GET /services/org.openmobilealliance.pres-content/users/sip:mamadou@micromethod.com/oma_status-
icon/rcs_status_icon HTTP/1.1
Host: doubango.org:8080
Connection: Keep-Alive
User-Agent: XDM-client/OMA1.1
X-3GPP-Intended-Identity: sip:bob@doubango.org
Pragma: No-Cache
Content-Type: application/vnd.oma.pres-content+xml
```

17 Signaling Compression (SigComp)

tinySigComp is a complete and compliant SigComp stack used to compress/decompress SIP messages over TCP, TLS, UDP and IPSec networks. Like all other stacks (DNS, HTTP/HTTPS, XCAP, MSRP …), you can use the tinySigComp alone in your own SIP/IMS application.

As many operators have begun to commercially deploy 3GPP IMS, the relevance of using SigComp to lower bandwidth usage will come quickly. Most operators (especially those using RCS and/or 3GPP LTE) will question how to reduce SIP signaling (registration, billing, presence, messaging …) bandwidth usage (who will pay bits?).
These questions will especially concern using SIP (or all other text-based protocols) in wireless handsets as part of 2.5G, 3GP and 4G cellular networks.

SigComp stands for Signaling Compression and has been defined in RFC 3320 by the Internet Engineering Task Force (IETF) ROHC working group.



Architecture

Many application protocols used for multimedia communications are text-based and engineered for bandwidth rich links. As a result the messages have not been optimized in terms of size. For example, typical IMS/SIP messages range from a few hundred bytes up to two thousand bytes or more. For this reason, SigComp is mandatory for 3GPP IMS and LTE networks.

SigComp could also be useful for RCS (Rich Communication Suite) networks because of the size of the SIP packets (more than three thousand bytes for presence publication). Using SigComp in IMS/RCS context will reduce the round-trip over slow radio links.

The stack implements:

□ RFC 3320: Signaling Compression (SigComp)
□ RFC 3321: Signaling Compression (SigComp) - Extended Operations
□ RFC 3485: The Session Initiation Protocol (SIP) and Session Description Protocol (SDP) Static

Dictionary for Signaling Compression (SigComp)
□ RFC 4077: A Negative Acknowledgement Mechanism for Signaling Compression
□ RFC 4464: Signaling Compression (SigComp) Users' Guide
□ RFC 4465: Signaling Compression (SigComp) Torture Tests
□ RFC 4896: Signaling Compression (SigComp) Corrections and Clarifications
□ RFC 5049: Applying Signaling Compression (SigComp) to the Session Initiation Protocol (SIP)
□ RFC 5112: The Presence-Specific Static Dictionary for Signaling Compression (Sigcomp)
□ RFC 1662: PPP in HDLC-like Framing
□ RFC 1951: DEFLATE Compressed Data Format Specification version
□ RFC 3174: US Secure Hash Algorithm 1 (SHA1)
□ 3GPP TR23.979 Annex C: Required SigComp Performance

17.1 Initialization

17.1.1 Parameters

SigComp parameters (RFC 3320 section 3.3) are used to configure the UDVM and change its capabilities, each offering a different amount of working memory, processing power etc.. In order to support this wide variation in endpoint capabilities, the following parameters are provided to modify SigComp behavior when receiving SigComp messages:

- decompression_memory_size
- state_memory_size
- cycles_per_bit
- SigComp_version

The code below shows how to configure these parameters.

```
tcomp_manager_setDecompression_Memory_Size(manager, 8192);
tcomp_manager_setCycles_Per_Bit(manager, 64);
tcomp_manager_setState_Memory_Size(manager, 8192);
```

By default, the SigComp version is equal to 2.0.

17.1.2 Dictionaries

Dictionaries are used to improve the compression ratio. By default, the stack supports both SIP/SDP and Presence dictionaries as per RFC 3485 and RFC 5112. These dictionaries should be explicitly added to the SigComp manager (not loaded by default) before to start compressing or decompressing any message.

The code below shows how to add these two dictionaries.

```
tcomp_manager_addSipSdpDictionary(manager);
tcomp_manager_addPresenceDictionary(manager);
```

17.1.3 Compressors

The SigComp stack offers possibilities to add your own compressors. By default, there are natively a dummy compressor and a deflate compressor. You can register up to `TCOMP_MAX_COMPRESSORS` (which is equal to 5) compressors. The deflate compressor implements RFC 1951 and is ready to be used (doesn't need to be registered) and is used as default.

The dummy compressor is only used when the dispatcher fails to compress using all other compressors. The dummy compressor produces a bytecode that simply instructs the decompressor to output the entire message (effectively sending it uncompressed, but within a SigComp message). For more information, please refer to RFC 4896 section 11.

A compressor is like an interface and should implements `tcomp_compressor_compress_f`, which is defined like this:

```
typedef tsk_bool_t (*tcomp_compressor_compress_f)(tcomp_compartment_t *lpCompartment, const void
*input_ptr, size_t input_size, void *output_ptr, size_t *output_size, tsk_bool_t stream);
```

The code below shows how to implements your own compressor.

```
tsk_bool_t my_compressor_compress (tcomp_compartment_t *lpCompartment, const void *input_ptr,
size_t input_size, void *output_ptr, size_t *output_size, tsk_bool_t stream)
{

       return tsk_true;

}
```

The code below shows how to add your own compressor.

```
int ret;

if((ret = tcomp_manager_addCompressor(manager, my_compressor_compress))){
       TSK_DEBUG_ERROR("Failed to add the compressor.");
}
```

You should use `tcomp_manager_removeCompressor()` to remove a compressor.

17.2 UDP Compression

17.3 UDP Decompression

17.4 TCP Compression

17.5 TCP Decompression

18 RObust Header Compression (ROHC)

3GPP TS 36.323

IETF RFC 3095

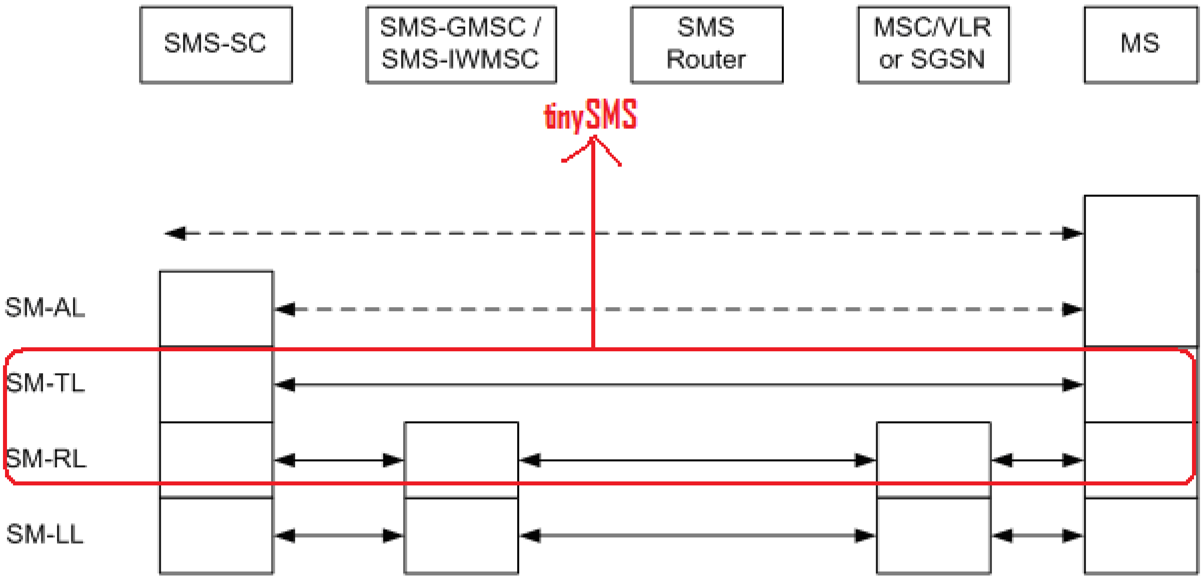
IETF RFC 4815

19 SMS over IP

SMS stands for Short Message Service or Silent Messaging Service and is a communication service standardized in the GSM mobile communication system, using standardized communications protocols allowing the interchange of short text messages between mobile telephone devices.

SMS technology has been specified by the ETSI in GSM 03.40 and 03.38 documents (3GPP TS 23.040 and 3GPP TS 23.038 respectively). These documents only describe how to use SMS over mobile networks (nothing for IP networks). The support of SMS over IP networks has been defined in 3GPP TS 24.341, which is fully implemented by doubango through tinySMS. To have access to all SMS functions, you should #include `"tinysms.h"` file in your project.

In this section, we will explain how to use SMS technology over IP within the IP Multimedia (IM) Core Network (CN) subsystem based on the Session Initiation Protocol (SIP) and SIP Events as defined in 3GPP TS 24.229.



Protocol layer overview for the Short Message Service

19.1 Modes

In real world, there are two ways to receive or send SMS messages over mobile networks: Binary (PDU) and Text mode.

19.1.1 Text mode

The UE shall send the SMS as any other SIP MESSAGE request and it's up to the IP-SM-GW to encode the payload before retransmission. This mode is out of scope because there is nothing special to do.

19.1.2 Binary mode

The UE shall implement the roles of an SM-over-IP sender and an SM-over-IP receiver, according the procedures in sections 5.3.1 and 5.3.2 in 3GPP TS 24.341.

The IMS core network shall take the role of an IP-SM-GW and support the general procedures in Section 5.3.3.1 of 3GPP TS 24.341, and the functions (answering of routing information query, and transport layer interworking) according to the procedures in Sections 5.3.3.3 and 5.3.3.4 in 3GPP TS 24.341.

It's up to the UE to encode the payload in binary format before sending the SMS. The payload contains a sequence of hexadecimal octets or decimal semi-octets strings encoded in binary format.

The overall PDU (Protocol Data Unit) string contains some useful information (SMSC address, Service Center Time Stamp, Sender Number, Message Reference ...) plus the actual message (payload).

The length of the payload can be up to 160 characters where each character represent 7bits [160/7bits], [140/8bits] or [70/16bits]. By default, each character represents 7bits encoded as per 3GPP TS 23.038.

For IMS and LTE Networks, SMS messages shall be encapsulated in RPDUs (Relay Protocol Data Unit) data string as defined in 3GPP TS 24.011 section 7.3. The RPDU data is transferred from SM entity to SM entity using SIP MESSAGE requests. These SIP requests shall use the MIME type "`application/vnd.3gpp.sms`" for this purpose.

19.2 Service provided by the SM-TL

The Short Message Transfer Layer (SM-TL) provides a service to the Short Message Application Layer (SM-AL). This service enables the SM-AL to transfer short messages to its peer entity, receive short messages from its peer entity and receive reports about earlier requests for short messages to be transferred.

19.2.1 SMS-SUBMIT

SMS-SUBMIT messages are used to convey short messages from the MS (Mobile Station) to the SC (Service Center). SMS-SUBMIT-REPORT messages are used for positive or negative acknowledgement to an SMS-DELIVER or SMS-STATUS-REPORT.

For more information, please refer to 3GPP TS 23.040 section 9.2.2.2.

The code below shows how to send a SMS-SUBMIT message from the MS to the SC (MO).

```c
#include "tsk.h"
#include "tinysms.h"

int ret;
tsms_tpdu_submit_t* submit = tsk_null;
tsk_buffer_t* buffer = tsk_null;
char* hex;
const char* smsc = "+331000009";
const char* destination = "+333361234567";
const char* short_message = "hello world";
uint8_t mr = 0xF5;

submit = tsms_tpdu_submit_create(mr, smsc, destination);

/* encode the user data to GSM 7-bit alphabet */
if((buffer = tsms_pack_to_7bit(short_message))){
	ret = tsms_tpdu_submit_set_userdata(submit, buffer, tsms_alpha_7bit);
	if((hex = tsms_tpdu_submit_tohexastring(submit))){
		TSK_DEBUG_INFO("SMS-SUBMIT=%s", hex);
		TSK_FREE(hex);
	}
	TSK_OBJECT_SAFE_FREE(buffer);

}

TSK_OBJECT_SAFE_FREE(submit);
```

The code above will output:

069133010000F911F50C913333163254760000AA0BE8329BFD06DDDF723619

069133010000F9	SMSC Address information (3GPP TS 23.040 section 9.1.2.5) 06: address length (octets) 91: type of the number is international and the numbering plan id is ISDN. 33010000F9: SMSC address, swapped and in semi octets (+331000009). As the SMSC address length is odd(9), then a trailing F has been added before swapping.
11	SMS-SUBMIT first octet (3GPP TS 23.040 section 9.2.2.2) Do not reject duplicates / Relative validity period format / No status report requested / No header in the user data / No reply path

F5	TP-Message-Reference (3GPP TS 23.040 section 9.2.3.6)
0C91333316325476	TP-Destination-Address (3GPP TS 23.040 section 9.2.3.8) 0C: The length of the destination addresses (semi octets). 91: type of the number is international and the numbering plan id is ISDN. 333316325476: destination address, swapped and in semi octets (+333361234567).
00	TP-Protocol-Identifier (3GPP TS 23.040 section 9.2.3.9)
00	TP-Data-Coding-Scheme (3GPP TS 23.040 section 9.2.3.10) Class 0 / GSM 7 bit default alphabet For more information about this field, please refer to 3GPP TS 23.038 section 4
AA	TP-Validity-Period (3GPP TS 23.040 section 9.2.3.12) 4 days
0B	TP-User-Data-Length (3GPP TS 23.040 section 9.2.3.16) 11: As we are using GSM 7bit default alphabet, then the length is the number of septets.
E8329BFD06DDDF723619	TP-User-Data (3GPP TS 23.040 section 9.2.3.24) 8-bit octets representing 7-bit data. The original data (8-bit) is “hello world”

The message should be sent over the network as binary content. For SMS-SUBMIT messages, binary serialization is performed by `tsms_tpdu_submit_serialize()`.

The code below shows how to serialize a SMS-SUBMIT message as binary content.

```
if ((buffer = TSK_BUFFER_CREATE_NULL())){
        if(!(ret = tsms_tpdu_submit_serialize(submit, buffer))){
            // ret = send(socket, buffer->data, buffer->size);
        }
        TSK_OBJECT_SAFE_FREE(buffer);
}
```

A SC receiving binary SMS-SUBMIT message (or any other SMS-*) from the network should use `tsms_tpdu_message_deserialize_mo()` function to deserialize the content.

19.2.2 SMS-DELIVER

SMS-DELIVER messages are used to convey short messages from the SC (Service Center) to the MS (Mobile Station). SMS-DELIVER-REPORT messages are used for positive or negative acknowledgement to an SMS-DELIVER or SMS-STATUS-REPORT.

For more information, please refer to 3GPP TS 23.040 section 9.2.2.1.

The code below shows how to receive a SMS-DELIVER message, sent from the SC to the MS (MT).

```
#include "tsk.h"
#include "tinysms.h"

tsms_tpdu_message_t* sms_any = tsms_tpdu_message_deserialize_mt(data, size);
if(sms_any && sms_any->mti == tsms_tpdu_mti_deliver_mt){
      //tsms_tpdu_deliver_t* sms_deliver = TSMS_TPDU_DELIVER(sms_any); ==> Yes we can !
      char* ascii;
      if((ascii = tsms_tpdu_message_get_payload(sms_any))){
              TSK_DEBUG_INFO("Message=%s", ascii);
              TSK_FREE(ascii);
      }
}
TS_OBJECT_SAFE_FREE(sms_any);
```

When the MS receive a SMS-DELIVER message, it should send back a SMS-DELIVER-REPORT message. The code bellow shows how to send this kind of message.

```
#include "tsk.h"
#include "tinysms.h"

int ret;
tsms_tpdu_report_t* report = tsk_null;
tsk_buffer_t* buffer = tsk_null;
const char* smsc = "+331000009";
tsk_bool_t isSUBMIT = tsk_false;
tsk_bool_t isERROR = tsk_false;

report = tsms_tpdu_report_create(smsc, isSUBMIT, isERROR);
buffer = TSK_BUFFER_CREATE_NULL();
if(!(ret = tsms_tpdu_report_serialize(report, buffer))){
        //send(buffer->data, buffer->size);

}
```

```
TSK_OBJECT_SAFE_FREE(report);
TSK_OBJECT_SAFE_FREE(buffer);
```

19.2.3 SMS-STATUS-REPORT

SMS-STATUS-REPORT messages are used to convey status reports from the SC (Service Center).to the MS (Mobile Station).

For more information, please refer to 3GPP TS 23.040 section 9.2.2.3.

The code below shows how to receive a SMS-STATUS-REPORT message, sent from the SC to the MS (MT).

```
#include "tsk.h"
#include "tinysms.h"

tsms_tpdu_message_t* sms_any = tsms_tpdu_message_deserialize_mt(buffer->data, buffer->size);
if(sms_any && sms_any->mti == tsms_tpdu_mti_status_report_mt){
      tsms_tpdu_status_report_t* sms_status_report = TSMS_TPDU_STATUS_REPORT(sms_any);
      switch(sms_status_report->st){
              case tsms_tpdu_status_received:
              case tsms_tpdu_status_forwarded:
              case tsms_tpdu_status_replaced:
              // ...
              default:
                      break;
      }
}
```

19.2.4 SMS-COMMAND

SMS-COMMAND messages are used to convey commands from the MS (Mobile Station) to the SC (Service Center).

For more information, please refer to 3GPP TS 23.040 section 9.2.2.4.

The code below shows how to send a SMS-COMMAND (DELETE) from the MS to the SC.

```
#include "tsk.h"
#include "tinysms.h"

tsms_tpdu_command_t* command = tsk_null;
char* hex;
tsk_buffer_t* buffer = tsk_null;
const char* smsc = "+331000009";
const char* destination = "+333361234567";
uint8_t mr = 0xF5;
uint8_t message_number = 0xF8;

command = tsms_tpdu_command_create(mr, smsc, destination, message_number, tsms_tpdu_cmd_delete);

if((hex = tsms_tpdu_command_tohexastring(command))){
      TSK_DEBUG_INFO("SMS-COMMAND=%s", hex);
      TSK_FREE(hex);
}
TSK_OBJECT_SAFE_FREE(command);
```

The code above will output:

069133010000F902F50002F80C9133331632547600

069133010000F9	SMSC Address information (3GPP TS 23.040 section 9.1.2.5) 06: address length (octets) 91: type of the number is international and the numbering plan id is ISDN. 33010000F9: SMSC address, swapped and in semi octets (+331000009). As the SMSC address length is odd(9), then a trailing F has been added before swapping.
02	SMS-COMMAND first octet (3GPP TS 23.040 section 9.2.2.4) No status report requested / No header in the user data
F5	TP-Message-Reference (3GPP TS 23.040 section 9.2.3.6)
00	TP-Protocol-Identifier (3GPP TS 23.040 section 9.2.3.9)
02	TP-Command-Type (3GPP TS 23.040 section 9.2.3.19) 0x02: Delete previously submitted short message.

F8	TP-Message-Number (3GPP TS 23.040 section 9.2.3.18)
0C91333316325476	TP-Destination-Address (3GPP TS 23.040 section 9.2.3.8) 0C: The length of the destination addresses (semi octets). 91: type of the number is international and the numbering plan id is ISDN. 333316325476: destination address, swapped and in semi octets (+333361234567).
00	TP-User-Data-Length (3GPP TS 23.040 section 9.2.3.16) 00: No data

The message should be sent over the network as binary content. For SMS-COMMAND messages, binary serialization is performed by `tsms_tpdu_command_serialize()`.

The code below shows how to serialize a SMS-COMMAND message as binary content.

```
if ((buffer = TSK_BUFFER_CREATE_NULL())){
        ret = tsms_tpdu_command_serialize(command, buffer);
        // ret = send(socket, buffer->data, buffer->size);

        TSK_OBJECT_SAFE_FREE(buffer);
}
```

A SC receiving binary SMS-COMMAND message (or any other SMS-*) over the network should use `tsms_tpdu_message_deserialize_mo() function` to deserialize the content.

19.3 Service provided by the SM-RL

The Short Message Relay Layer (SM-RL) provides a service to the Short Message Transfer Layer (SM-TL). This service enables the SM-TL to send Transfer Protocol Data Units (TPDUs) to its peer entity, receive TPDUs from its peer entity and receive reports about earlier requests for TPDUs to be transferred.

For more information about how doubango provides SM-TL services, please refer to the previous section.

19.3.1 RP-DATA

RP-DATA messages are use to relay the TPDUs. These messages could be sent from the Network to Mobile Station (MT) or from the Mobile Station to the Network (MO).

For more information, please refer to 3GPP TS 24.011 section 7.3.1.

The code below shows how to send a RP-DATA message with a RP-User-Data (see 3GPP TS 23.011 section 8.2.5.3) information element which includes SMS-SUBMIT as type indicator (this use case comes from 3GPP TS 24.341 chapter B.5).

```
#include "tsk.h"
#include "tinysms.h"

int ret;
tsk_buffer_t* buffer = tsk_null;
tsms_tpdu_submit_t* sms_submit = tsk_null;
tsms_rpdu_data_t* rp_data = tsk_null;
const char* smsc = "+331000009";
const char* destination = "+333361234567";
const char* short_message = "hello world";
uint8_t mr = 0xF5;
char* hex;

// create Mobile Originated SMS-SUBMIT message
sms_submit = tsms_tpdu_submit_create(mr, smsc, destination);
// Set content for SMS-SUBMIT
if((buffer = tsms_pack_to_7bit(short_message))){
       ret = tsms_tpdu_submit_set_userdata(sms_submit, buffer, tsms_alpha_7bit);
       TSK_OBJECT_SAFE_FREE(buffer);
}
// create RP-DATA(SMS-SUBMIT) message and print its content (for test only)
rp_data = tsms_rpdu_data_create_mo(mr, smsc, TSMS_TPDU_MESSAGE(sms_submit));
if((hex = tsms_rpdu_message_tohexastring(TSMS_RPDU_MESSAGE(rp_data)))){
       TSK_DEBUG_INFO("RP-DATA=%s", hex);
       TSK_FREE(hex);
}

TSK_OBJECT_SAFE_FREE(buffer);
TSK_OBJECT_SAFE_FREE(sms_submit);
TSK_OBJECT_SAFE_FREE(rp_data);
```

The code above will output:

00F500069133010000F91F069133010000F911F50C913333163254760000AA0BE8329BFD06DDDF723619

00	RP-Message Type (3GPP TS 24.011 section 8.2.2) 00: RP-DATA (Mobile station to Network). Mobile Originated.
F5	RP-Message Reference (3GPP TS 24.011 section 8.2.3) F5: See the code.
00	RP-Originator Address (3GPP TS 24.011 section 8.2.5.1) 00: For Mobile originated messages, this field is empty.
069133010000F9	RP-Destination Address (3GPP TS 23.011 section 8.2.5.2) 06: address length (octets) 91: type of the number is international and the numbering plan id is ISDN. 33010000F9: SMSC address, swapped and in semi octets (+331000009). As the SMSC address length is odd(9), then a trailing F has been added before swapping. For Mobile originated messages, this field contains the SMSC address.
11F50C913333163254760000AA0BE8329BFD06DDDF723619	RP-User Data (3GPP TS 24.011 section 8.2.5.3) 11: Length indicator F50C913333163254760000AA0BE8329BFD06DDDF723619: SMS-SUBMIT message. For more information, please refer to the SMS-SUBMITsection.

The message should be sent over the network as binary content. For RP-DATA messages, binary serialization is performed by `tsms_rpdu_data_serialize()`.

The code below shows how to serialize a RP-DATA message as binary content.

```
if ((buffer = TSK_BUFFER_CREATE_NULL())){
        ret = tsms_rpdu_data_serialize(rp_data, buffer);
        // ret = send(socket, buffer->data, buffer->size);

        TSK_OBJECT_SAFE_FREE(buffer);
}
```

A SC receiving binary RP-DATA message (or any other RP-*) over the network should use `tsms_rpdu_message_deserialize()` function to deserialize the content.

The code below shows how to receive a RP-DATA message with a RP-User-Data (see 3GPP TS 23.011 section 8.2.5.3) information element which includes SMS-DELIVER as type indicator (this use case comes from 3GPP TS 24.341 chapter B.6).

```
#include "tsk.h"
#include "tinysms.h"

tsms_rpdu_message_t* rp_message = tsk_null;
tsms_tpdu_message_t* tpdu = tsk_null;

if(!(rp_message = tsms_rpdu_message_deserialize(data, size))){ /* Deserialize RP-* Message */
       TSK_DEBUG_ERROR("Failed to deserialize the RP-MESSAGE");
       goto bail;
}

if(rp_message->mti == tsms_rpdu_type_data_mt){ /* Mobile Terminated RP-DATA*/
       char* ascii = tsk_null;
       tsms_rpdu_data_t* rp_data = TSMS_RPDU_DATA(rp_message);
       if((tpdu = tsms_tpdu_message_deserialize_mt(rp_data->udata->data, rp_data->udata->size))){
/* Deserialize SMS-* Message (From SC to MS) */
              if(tpdu->mti == tsms_tpdu_mti_deliver_mt){ /* Mobile Terminated SMS-DELIVER*/
                     if((ascii = tsms_tpdu_message_get_payload(tpdu))){
                            TSK_DEBUG_INFO("ASCII message=%s", ascii);
                            TSK_FREE(ascii);
                     }
              }
       }
}

bail:
TSK_OBJECT_SAFE_FREE(rp_message);
TSK_OBJECT_SAFE_FREE(tpdu);
```

19.3.2 RP-SMMA

RP-SMMA messages are sent by the mobile station to relay a notification to the network that the mobile has memory available to receive one or more short messages.

For more information, please refer to 3GPP TS 24.011 section 7.3.2.

The code below shows how to send a RP-SMMA message.

```
#include "tsk.h"
#include "tinysms.h"

int ret;
tsk_buffer_t* buffer = tsk_null;
tsms_rpdu_smma_t* rp_smma = tsk_null;
uint8_t mr = 0xF5;

// create RP-SMMA message
rp_smma = tsms_rpdu_smma_create(mr);
// serialize
buffer = TSK_BUFFER_CREATE_NULL();
ret = tsms_rpdu_data_serialize(rp_smma, buffer);
// send(socket, buffer->data, buffer->size);

TSK_OBJECT_SAFE_FREE(buffer);
TSK_OBJECT_SAFE_FREE(rp_smma);
```

19.3.3 RP-ACK

RP-ACK messages are sent between the MSC and the mobile station in both directions and used to relay the acknowledgement of a RP-DATA or RP-SMMA message reception.

For more information, please refer to 3GPP TS 24.011 section 7.3.2.3.

The code below shows how to send a RP-ACK message with a RP-User-Data (see 3GPP TS 23.011 section 8.2.5.3) information element which includes SMS-DELIVER-REPORT as type indicator (this use case comes from 3GPP TS 24.341 chapter B.6 section 8).

```
#include "tsk.h"
#include "tinysms.h"

int ret;
tsk_buffer_t* buffer = tsk_null;
tsms_tpdu_report_t* sms_report = tsk_null;
tsms_rpdu_ack_t* rp_ack= tsk_null;
const char* smsc = "+331000000";
tsk_bool_t isSUBMIT = tsk_false; /* is SMS-DELIVER-REPORT */
tsk_bool_t isERROR = tsk_false;
uint8_t mr = 0xF5;

// create SMS-DELIVER-REPORT message
sms_report = tsms_tpdu_report_create(smsc, isSUBMIT, isERROR);
// create Mobile Originated RP-ACK message
rp_ack = tsms_rpdu_ack_create_mo(mr, TSMS_TPDU_MESSAGE(sms_report));
// serialize and send
buffer = TSK_BUFFER_CREATE_NULL();
if(!(ret = tsms_rpdu_data_serialize(rp_ack, buffer))){
        // send(socket, buffer->data, buffer->size);
}
TSK_OBJECT_SAFE_FREE(buffer);
TSK_OBJECT_SAFE_FREE(sms_report);
TSK_OBJECT_SAFE_FREE(rp_ack);
```

19.3.4 RP-ERROR

RP-ERROR messages are sent between the MSC and the mobile station in both directions and used to relay an error cause from an erroneous short message or notification transfer attempt.

For more information, please refer to 3GPP TS 24.011 section7.3.2.4.

The code below shows how to send a RP-ERROR message with a RP-User-Data (see 3GPP TS 23.011 section 8.2.5.3) information element which includes SMS-DELIVER-REPORT as type indicator. In this example, the error message is sent because the “call is barred”. For more information about the cause values that may be contained in an RP-ERROR message, please refer to 3GPP TS 24.011 section 8.2.5.4.

```
#include "tsk.h"
#include "tinysms.h"

int ret;
tsk_buffer_t* buffer = tsk_null;
tsms_tpdu_report_t* sms_report = tsk_null;
tsms_rpdu_error_t* rp_error= tsk_null;
tsk_bool_t isSUBMIT = tsk_false; /* is SMS-DELIVER-REPORT */
tsk_bool_t isERROR = tsk_true;
const char* smsc = "+331000000";
uint8_t mr = 0xF5;

// create SMS-DELIVER-REPORT message
sms_report = tsms_tpdu_report_create(smsc, isSUBMIT, isERROR);
// create Mobile Originated RP-ERROR message
rp_error = tsms_rpdu_error_create_mo(mr, TSMS_TPDU_MESSAGE(sms_report), 0x0A/*call barred*/);
// serialize
buffer = TSK_BUFFER_CREATE_NULL();
if(!(ret = tsms_rpdu_data_serialize(rp_error, buffer))){
      // send(socket, buffer->data, buffer->size);
}

TSK_OBJECT_SAFE_FREE(buffer);
TSK_OBJECT_SAFE_FREE(sms_report);
TSK_OBJECT_SAFE_FREE(rp_error);
```

20 MSRP

21 SDP

22 SIP (3GPP IMS/LTE)

22.1 Initialization

In this chapter we will explain the minimal parameters to set in order to initialize the stack and how to start it.

As the IMS/LTE stack depends on the network library (tinyNET), you MUST call tnet_startup() before using any SIP function (`tsip_*`). tnet_cleanup() is used to terminate use of network functions. Before starting the stack, you must set: **the realm and the user's IMPI and IMPU**. For more information about these fields, please refer to the next sections.

To have access to all SIP functions, you should #include `"tinysip.h"` file in your project.

22.1.1 Realm

The realm is the name of the domain name to authenticate to. It should be a valid SIP URI (e.g. sip:open-ims.test).

The realm is mandatory and should be set before the stack starts. You should never change its value once the stack is started. If the address of the Proxy-CSCF is missing, then the stack will automatically use DNS NAPTR+SRV and/or DHCP mechanisms for dynamic discovery. The value of the realm will be used as domain name for the DNS NAPTR query. For more information about how to set the Proxy-CSCF IP address and port, please refer to section 22.1.8.

22.1.2 IMS Private Identity (IMPI)

The IMPI is a unique identifier assigned to a user (or UE) by the home network. It could be either a SIP URI (e.g. sip:bob@open-ims.test), a tel URI (e.g. tel:+33100000) or any alphanumeric string (e.g. bob@open-ims.test or bob). It is used to authenticate the UE (username field in SIP Digest Authorization/Proxy-Authorization header).

In the real word, it should be stored in an UICC (Universal Integrated Circuit Card).

For those using doubango as a basic (IETF) SIP stack, the IMPU should coincide with their authentication name.

The IMPI is mandatory and should be set before the stack starts. You should never change the IMPI once the stack is started.

22.1.3 IMS Public Identity (IMPU)

As its name says, it's you public visible identifier where you are willing to receive calls or any demands. An IMPU could be either a SIP or tel URI (e.g. tel:+33100000 or sip:bob@open-ims.test). In IMS world, a user can have multiple IMPUs associated to its unique IMPI.

For those using doubango as basic SIP stack, the IMPU should coincide with their SIP URI (a.k.a SIP address).

The IMPU is mandatory and should be set before the stack starts. You should never change the IMPU once the stack is started (instead, change the P-Preferred-Identity if you want to change your default public identifier).

22.1.4 Preferred Identity

As a user has multiple IMPUs, it can for each outgoing request, defines which IMPU to use by setting the preferred identity. The user should check that this IPMU is not barred. An IMPU is barred if it doesn't appear in the associated URIs returned in the 200 OK REGISTER.

By default, the preferred identity is the first URI in the list of the associated identities. If the IMPU used to REGISTER the user is barred, then the stack will use the default URI returned by the S-CSCF.

You should never manually set this SIP header (P-Preferred-Identity); it’s up to the stack.

22.1.5 Creating the stack

`tsip_stack_create(callback, realm_uri, impi_uri, impu_uri, …)` function is used to create a new 3GPP IMS/LTE stack. This function will return a valid handle if succeed and `Null` otherwise.

In the code below, we only use mandatory parameters (except the `password`). The Proxy-CSCF IP address and Port are missing and you should use `TSIP_STACK_SET_PROXY_CSCF()` to set them if required. For more information, please refer to the next section (22.1.8).

```
tsip_stack_handle_t *stack = tsk_null;
int ret;

const char* realm_uri = "sip:open-ims.test";
const char* impi_uri = "bob@open-ims.test";
const char* impu_uri = "sip:bob@open-ims.test";

int app_callback(const tsip_event_t *sipevent)
{
       switch(sipevent->type){
              case tsip_event_register:
                      {      /* REGISTER */
                             break;
                      }
              case tsip_event_invite:
                      {      /* INVITE */
                             break
                      }
              case tsip_event_message:
                      {      /* MESSAGE */
                             break
                      }
              case tsip_event_publish:
                      { /* PUBLISH */
                             break
                      }
              case tsip_event_subscribe:
                      {      /* SUBSCRIBE */
                             break
                      }
              case tsip_event_transporterr:
                      {      /* Transport error */
                             break
                      }
              case tsip_event_terminated:
                      {      /* Session (SIP dialog) terminated */
                             break
                      }
              /* case …*/
              /* case …*/
              /* case …*/
              default:
                      {      /* Unsupported */
                             TSK_DEBUG_WARN("%d not supported as SIP event.", sipevent->type);
                             return -3;
                      }
       }
       return 0;
}

// ...
tnet_startup();

// ...

stack = tsip_stack_create(app_callback, realm_uri, impi_uri, impu_uri,
       TSIP_STACK_SET_PASSWORD("mysecret"),
       // ...other macros...
       TSIP_STACK_SET_NULL());

// ...
```

```
TSK_OBJECT_SAFE_FREE(stack);

tnet_cleanup();
```

`TSIP_STACK_SET_NULL()` macro is mandatory, it's used to terminate the list of parameters passed to the function. This macro should always be the last one. For more information, please refer to the SIP API reference.

`TSIP_STACK_SET_PASSWORD()` macro is not mandatory, it's used to set the user's password. Credentials are always used at stack-level. The callback function will be used to alert the user about new incoming SIP messages, media state changes, errors … More information about the callback function is given below.

A stack is a well-defined object and should be destroyed using `TSK_OBJECT_SAFE_FREE()`. When destroyed, the stack automatically frees all underlying sessions and cleanly hang-up all SIP dialogs.

22.1.6 Starting the stack

`tsip_stack_start(stack)` function is used to start the stack. This function return zero if succeed and non-zero error code otherwise. If the stack fails to start, you should look at the debug output for more information. You can call this function in the UI thread as it will immediately return and all additional tasks will be done in a separate thread. When user's operations are performed on the stack while it's starting in background, theses operations will be queued and executed when the stack starts.

To stop the stack, use `tsip_stack_stop(stack)`. This function is thread-safe and should return zero if succeed (or if the stack is already stopped) and non-zero error code otherwise.

22.1.7 Callback function

The callback function is not mandatory but it's highly recommended to provide one. This function is called by the stack to alert the user end application about both SIP and media events.

The signature of the callback function is given below:

```
 int app_callback(const tsip_event_t *sipevent);
```

The end user application should return zero for success and non-zero error code otherwise.

22.1.8 Setting the Proxy-CSCF

You should set the Proxy-CSCF address and IP only if required. Dynamic discovery mechanisms (DNS NAPTR and/or DHCP) should be used.

The code below shows how to set the Proxy-CSCF IP address and Port. If the port is missing, then its default value will be 5060.

```
unsigned pcscf_port = 4060;
const char* pcscf_ip = "6666::eeee:bbb:ccc:ddd";
const char* transport = "udp"; /* 'udp' or 'tcp' or 'tls' or 'sctp' */
const char* ip_version = "ipv6"; /* 'ipv6' or 'ipv4' or 'ipv6/ipv4' or 'ipv4/ipv6' */

tsip_stack_handle_t *stack = tsip_stack_create(test_stack_callback, realm_uri, impi_uri, impu_uri,

      TSIP_STACK_SET_PROXY_CSCF(pcscf_ip, pcscf_port, transport, ip_version),

      TSIP_STACK_SET_NULL());
```

You can also use `tsip_stack_set()` to set these values. The Proxy-CSCF IP address and port should never be changed once the stack is started.

22.1.9 Setting the local IP and Port

The code below shows how to set the local IP address and port. This option is useful for NAT traversal.

On Google Android systems, on the emulator, the stack could fail to retrieve the local IP address. A workaround is to set the IP address yourself. The IP address of the emulator is always "10.0.2.15". For all other systems (Windows XP/7/CE, µLinux, MAC OS X, iPhone, Symbian …) this function is useless and you should let the stack select the best local IP address and Port.

```
unsigned local_port = 1234;
const char* local_ip = "5555::aaaa:bbb:ccc:ddd";

unsigned pcscf_port = 4060;
const char* pcscf_ip = "6666::eeee:bbb:ccc:ddd";
const char* transport = "udp"; /* 'udp' or 'tcp' or 'tls' or 'sctp' */
const char* ip_version = "ipv6"; /* 'ipv6' or 'ipv4' or 'ipv6/ipv4' or 'ipv4/ipv6' */

tsip_stack_handle_t *stack = tsip_stack_create(test_stack_callback, realm_uri, impi_uri, impu_uri,

      TSIP_STACK_SET_LOCAL_IP(local_ip),
      TSIP_STACK_SET_LOCAL_PORT(local_port),

      TSIP_STACK_SET_PROXY_CSCF(pcscf_ip, pcscf_port, transport, ip_version),

      TSIP_STACK_SET_NULL());
```

You can also use tsip_stack_set() to set these values. The local IP address and port should never be changed once the stack is started.

The family (IPv4 or IPv6) of the local IP address must be the same as the Proxy-CSCF.

22.1.10 Headers

Stack-level headers will be added to all outgoing requests. There are three levels: stack-level, session-level and action-level (a.k.a request-level). For the more information about the session and action (or request) levels, please refer to the next sections.

All stack-level headers should be set by using TSIP_STACK_SET_HEADER() macro when the stack is about to be created (tsip_stack_create()) or later (tsip_stack_set()).

The code below shows how to set stack-level headers.

```
tsip_stack_set(stack,
      TSIP_STACK_SET_HEADER("User-Agent", "IM-client/OMA1.0 doubango/v1.0.0"),
      TSIP_STACK_SET_HEADER ("Allow", "INVITE, ACK, CANCEL, BYE, MESSAGE, OPTIONS, NOTIFY, PRACK,
UPDATE, REFER"),

      TSIP_STACK_SET_NULL());
```

To, From, Expires, Call-Id, CSeq, Contact, Via, Content-Length, Authorization and P-Preferred-Identity (there are probably more) headers should not be set using the former macro.

TSIP_STACK_UNSET_HEADER() and TSIP_STACK_SET_HEADER() macros are used to remove or update a previously added stack-level header, respectively.

By default, all stack-level SIP headers will be sent in compact format to reduce bandwidth usage. To change this behavior, set TSIP_COMPACT_HEADERS (tinysip_config.h or config.h) macro value to zero.

22.1.11 Options

Options (also known as "parameters") macros are used to configure the stack. For example, TSIP_STACK_SET_PROXY_CSCF() macro used to set the Proxy-CSCF address and port is part of these macros. They all start with TSIP_STACK_SET_*() and are passed to tsip_stack_create() or tsip_stack_set() functions.

The code below shows how to use these options to enable IPSec security agreement (as per RFC 3329) and to set user's preferences.

```
tsip_stack_set(stack,
              TSIP_STACK_SET_SECAGREE_IPSEC(tsk_true),
              TSIP_STACK_SET_IPSEC_PARAMS("hmac-md5-96", "des-ede3-cbc", "trans", "esp"),
              TSIP_STACK_SET_NULL());
```

For more information, please refer to the SIP API reference.

22.2 Sessions

A session could be seen as a SIP dialog as per RFC 3261 section 12. For example, REGISTER, SUBSCRIBE, PUBLISH, MESSAGE, INVITE, … are seen as sessions even if MESSAGE and PUBLISH do not really create a SIP dialog as per RFC 3261.

The code below shows how to create a session.

```
tsip_ssession_handle_t *session = tsip_ssession_create(stack,

       TSIP_SSESSION_SET_NULL());
```

The first parameter, `stack`, is the IMS/LTE stack created using `tsip_stack_create()`. For more information about how to create IMS/LTE stack, please refer to the previous sections.

`TSIP_SSESSION_SET_NULL()` macro, used as parameter, is manadatory and should always be the last one.

A session is a well-defined object and should be destroyed using `TSK_OBJECT_SAFE_FREE()`. As the session depends on the `stack`, it should be destroyed first.

22.2.1 Headers

Session-level headers will be added to all requests which belong to this same session (same Call-Id and To/From tags). For example, when used in the REGISTER session, these headers will be added to the initial register, all reregisters (refreshes) and deregister requests.

Session-level headers could be added when the session is about it be created (`tsip_ssession_create()`) or at any time (`tsip_ssession_set()`).

The code below shows how to add three headers to the session.

```
tsip_ssession_set(session,
       TSIP_SSESSION_SET_HEADER("Supported", "norefersub"),
        TSIP_SSESSION_SET_HEADER("Privacy", "header;id"),
       TSIP_SSESSION_SET_HEADER("P-Access-Network-Info", "3GPP-UTRAN-TDD;utran-cell-id-
3gpp=AAAAA0000BBBB"),

       TSIP_SSESSION_SET_NULL());
```

`TSIP_SESSION_UNSET_HEADER()` and `TSIP_SESSION_SET_HEADER()` macros are used to remove or update a previously added session-level header, respectively.

By default, all session-level SIP headers will be sent in compact format to reduce bandwidth usage. To change this behavior, set `TSIP_COMPACT_HEADERS` (`tinysip_config.h` or `config.h`) macro value to zero.

If a header added at stack-level is also added at session-level, then it will be duplicated.

22.2.2 Options

Session-level options are used to configure a session. You should always use this mechanism to configure the session instead of using SIP headers. For example, to set the "expires" value, you should use `TSIP_SSESSION_SET_EXPIRES(600000)` macro helper instead of `TSIP_SSESSION_SET_HEADER("Expires", "600000")`. This also applies to "To", "From" … headers.

The code below shows how to configure a session.

```
tsip_ssession_set(session,
       TSIP_SSESSION_SET_OPTION(TSIP_SSESSION_OPTION_EXPIRES, "600000"),

       TSIP_SSESSION_SET_OPTION(TSIP_SSESSION_OPTION_NO_CONTACT, tsk_false),
       TSIP_SSESSION_SET_OPTION(TSIP_SSESSION_OPTION_TO, "sip:alice@open-ims.test"),
        TSIP_SSESSION_SET_NULL());
```

Session-level options could be added when the session is about it be created (`tsip_ssession_create()`) or at any time (`tsip_ssession_set()`).

22.2.3 Capabilities

Capabilities are used to set the UE characteristics by using the "Contact" header. As the programmer doesn't have access to the contact header, `TSIP_SSESSION_SET_CAPS()` is used to advertise the UE caps. For more information, please refer to IETF RFC 3840. When the message don't have "Contact" header (e.g. MESSAGE or PUBLISH), then you should use the "Accept-Contact" header to set the capabilities.

Capabilities could be added when the session is about it be created (`tsip_ssession_create()`) or at any time (`tsip_ssession_set()`).

The code below shows how to add caps.

```
tsip_ssession_create(stack,

        TSIP_SSESSION_SET_CAPS("+g.oma.sip-im", ""),
        TSIP_SSESSION_SET_CAPS("+g.3gpp.smsip", ""),
        TSIP_SSESSION_SET_CAPS("automata", ""),
        TSIP_SSESSION_SET_CAPS("language", "\"en,fr\""),

        TSIP_SSESSION_SET_NULL());
```

This will result in this kind of Contact header:

```
Contact: <sip:bob@192.168.16.82:4455;transport=udp>;expires=600000;+g.oma.sip-im;
+audio;automata;language="en,fr"
```

You should use `TSIP_SSESSION_UNSET_CAPS()` macro to remove a previously added caps.

22.2.4 Outgoing

It's up to the caller to create and configure outgoing sessions. There is nothing special to do.

22.2.5 Incoming

It's up to the stack to create and configure incoming sessions.

22.3 Requests

A request is always associated to a session. For more information about how to create and configure sessions, please refer to the previous section.

Requests are commonly called "actions" and are always associated to a SIP method (e.g. REGISTER, SUBSCRIBE, PUBLISH, UPDATE, MESSAGE, REFER …). For example, to send a SIP REGISTER request, you should use `tsip_action_REGISTER()`. For more information about all available actions, please refer to the SIP API reference.

If it's the initial request, then the dialog will be created, otherwise the same dialog is reused (refresh). It's up to the stack to maintain the dialog by sending the periodic refreshes.

22.3.1 Headers

Request-level (a.k.a action-level) headers will be added to an initial outgoing request until a final response is received (non-1xx except 401/407/423). This means that if you add a header to an initial request, when challenged, the second request (with credentials) will also have this header. The same applies to 423 responses. However, if the first response is any other reliable response, then all subsequent requests using the same session won't have this header.

The code below shows how to send a SIP REGISTER request.

```
tsip_action_REGISTER(session,
        TSIP_ACTION_SET_HEADER("My-Header-1", "My-Value-1"),
        TSIP_ACTION_SET_HEADER("My-Header-2", "My-Value-2"),

        TSIP_ACTION_SET_NULL());
```

If a header added at stack-level or session-level is also added at request-level, then this header will be duplicated. The `session` parameter is a SIP session object created as per section `22.2 Sessions`.

22.3.2 Outgoing

22.3.3 Incoming

22.4 Proxy-CSCF discovery

22.4.1 DHCPv4

22.4.2 DHCPv6

22.4.3 DNS NAPTR+SRV

22.5 Registration

22.5.1 IMS-AKA

doubango fully implements IMS-AKA (Authentication and Key Agreement) as per 3GPP TS 24.229 and 3GPP TS 33.203.

In order to compute the response, the framework needs the IK, CK, AMF and Operator Id values. For more information about these fields, please refer to 3GPP TS 3G Security series (3GPP TS 35.205, 3GPP TS 35.206, 3GPP TS 35.207, 3GPP TS 35.208 and 3GPP TS 35.909).

The IK (Integrity Key) and CK (Cipher Key) values are automatically computed from the user's credentials and you don't need to supply them. However, it's up to you to set the AMF (Authentication Management Field) and Operator Id values.

The AMF field is a 16-bit (`uint16_t`) value which is used as an input to the functions f1 and f1*. You should use `TSIP_STACK_SET_IMS_AKA_AMF()` macro to set the AMF value.

The Operator Id filed is a 128-bit (16 bytes) value which is used as an input to the functions f1, f1*, f2, f3, f4, f5 and f5*. You should use `TSIP_STACK_SET_IMS_AKA_OPERATOR_ID()` macro to set the Operator Id value at stack-level

Ipsec, Sec-Agree (nego),

22.5.2 Initial registration

22.5.3 Early IMS Security

Basic SIP application should use this option to disable some heavy IMS authentication procedures. Early IMS as per 3GPP TS 33.978 is partially supported and should be set at stack-level by using `TSIP_STACK_SET_EARLY_IMS(ENABLED_BOOL)` macro.

22.5.4 Identities

Getting P-Associated-URIs (and checking if an identity is barred)

Setting P-Preferred-Identities

22.5.5 Network-Initiated de-registration

22.5.6 Network-initiated re-registration

22.6 Emergency Service

3GPP TS 23.167 (section 6.2 and annex H)

3GPP TS 23.41

22.7 Signalling Compression (SigComp)

22.8 RObust Header Compression (ROHC)

3GPP TS 36.323

IETF RFC 3095

IETF RFC 4815

22.9 Presence

22.9.1 Subscription

`tsip_action_SUBSCRIBE()` to subscribe and `tsip_action_UNSUBSCRIBE()` to unsubscribe.

22.9.2 Publication

`tsip_action_PUBLISH()` to publish (or refresh) your presence and `tsip_action_UNPUBLISH()` to unpublish your presence.

22.10 Pager Mode IM (MESSAGE)

OMA SIMPLE IM

22.11 Session Mode IM (MSRP)

OMA SIMPLE IM

22.12 File Transfer

OMA SIMPLE IM

22.13 Image Sharing

GSMA IR.79

22.14 SMS over IP

On sending a REGISTER request, the SM-over-IP receiver shall indicate its capability to receive traditional short messages over IMS network by including a "+g.3gpp.smsip" parameter into the Contact header according to RFC 3840 [16]. This is done like this:

```
tsip_ssession_set(session,
      TSIP_SSESSION_SET_CAPS("+g.3gpp.smsip", ""),
      TSIP_SSESSION_SET_NULL());
```

In this chapter, “PSI of the SC” defines the SMSC address (SIP or tel URI).

22.14.1 Sending SMS

Binary SMS messages are sent as per 3GPP TS 24.341 section 5.3.1.2.

According to this section, the SM-over-IP sender shall send a SIP MESSAGE encapsulating RPDU message encoded as per 3GPP TS 24.011 section 7.3. The SM-over-IP sender shall use MIME type “application/vnd.3gpp.sms” for this purpose.

22.14.2 Receiving SMS

22.14.3 Roaming Considerations

3GPP TS 23.272
3GPP TS 23.221
3GPP TS 24.301

22.15 NAT Traversal

22.15.1 rport

This is the simplest way to deal with NATs problem at signalling (SIP only) plan for connectionless protocols (e.g. UDP). **rport** has been defined in IETF RFC 3581 and apply to both connectionless (e.g. UDP) and connection-oriented (e.g. TCP, TLS, SCTP) protocols. This option is always enabled even if you use STUN, TURN or ICE (not exclusive).

Because connection-oriented protocols such as TCP are bidirectional it is easier to deal with NAT traversal (all responses will be sent back using the same connection from which the request has been received from).

The philosophy (of **rport**) is that, when you add this attribute (**rport**) in your requests (`Via: SIP/2.0/UDP 192.168.16.108:27208;branch=z9hG4bK1234;rport`) then all responses will be sent back to the `ip:port` from which the request has been received from. This parameter MUST be added in the top most “Via” header.

The response will contain a new parameter (“received”) containing the IP address from which the request has been received and the **rport** value will be filled with the mapped source port. In this way the client (the sender of the request) can learn it's routable IP address and port (`Via: SIP/2.0/UDP 192.168.16.108:27208;branch=z9hG4bK1234;rport=1234;received=10.1.1.1`).

By examining the response the client can know if it's behind a NAT or not.

This solution allows receiving responses, but doesn’t tackle the problem of incoming requests as our AOR is still incorrect. A workaround is to change the AOR by using `TSIP_STACK_SET_AOR()` macro helper.

The problem with this solution is that it only deals with SIP messages and cannot be used for RTP/RTCP packets as the SDP will contain wrong IP addresses and ports. STUN or ICE are used to deal with this issue. For more information, please refer to the next sections.

22.15.2 STUN

22.15.3 TURN

22.15.4 ICE

22.16 Supplementary Services (MMTel)

22.16.1 Originating Identification Presentation

3GPP TS 24.607

22.16.2 Terminating Identification Presentation

3GPP TS 24.608

22.16.3 Originating Identification Restriction

3GPP TS 24.607

22.16.4 Terminating Identification Restriction

3GPP TS 24.608

22.16.5 Communication Diversion Unconditional

3GPP TS 24.604

22.16.6 Communication Diversion on not Logged in 3GPP TS 24.604

22.16.7 Communication Diversion on Busy

3GPP TS 24.604

22.16.8 Communication Diversion on not Reachable

3GPP TS 24.604

22.16.9 Communication Diversion on No Reply

3GPP TS 24.604

22.16.10 Barring of All Incoming Calls

3GPP TS 24.611

22.16.11 Barring of All Outgoing Calls

3GPP TS 24.611

22.16.12 Barring of Outgoing International Calls

3GPP TS 24.611

22.16.13 Barring of Incoming Calls - When Roaming

3GPP TS 24.611

22.16.14 Communication Hold

3GPP TS 24.610

22.16.15 Message Waiting Indication

3GPP TS 24.606

22.16.16 Communication Waiting

3GPP TS 24.615

22.16.17 Ad-Hoc Multi Party Conference

3GPP TS 24.605

22.16.18 Supplementary Service Configuration

3GPP TS 24.623

22.17 SIP Preconditions (QoS)

22.18 Provisional Response Acknowledgements (PRACK)

22.19 Session Timers

22.20 DTMF

22.21 Torture Tests

IETF RFC 4475

23 Compilation

For all operating systems you should use GNU Build Tools (autoconf, automake, autoheader, configure….) and GNU libtool (for shared libraries) to compile doubango. However, there are some exceptions where it's easier to use other mechanisms (Borland, Visual Studio, Eclipse, Xcode, makefiles …), for each system, please refer to the next sections to see what we propose.

doubango V1.0.0 contains at least 15 projects (libraries) and they should be built in a predefined order. It's not mandatory to build them all. The table below shows each library with its functionalities and dependencies. These projects should be compiled in the order given below.

	Functionalities	**Dependencies**
tinySAK (tiny Swiss Army Knife) Prefix: `tsk`	➢ ANSI-C Object Programing ➢ Linked lists ➢ String utility functions ➢ Memory management ➢ Dynamic buffers ➢ Threading ➢ Runnable ➢ Mutexes ➢ Semaphores ➢ Conditional Variables ➢ Timers ➢ Time ➢ Final State Machine (FSM) manager ➢ Base64 encoder/decoder ➢ UUID generator ➢ CRC32 and CRC16 ➢ URL encoder/decoder ➢ SHA-1, MD5, HMAC-MD5, HMAC-SHA-1	No dependencies
tinyNET (Networking) Prefix: `tnet`	➢ IPv4/IPv6 Sockets (UDP, TCP, TLS and SCTP) ➢ DHCPv4/v6 ➢ DNS (NAPTR, PTR, SRV, MX, A, AAAA, OPT, CNAME ...) ➢ ENUM ➢ NAT Traversal (STUN, TURN and ICE)	tinySAK
tinyHTTP (HTTP/HTTPS stack) Prefix: `thttp`	➢ Digest/Basic Authentication ➢ Pipelining ➢ CONNECT, DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT and TRACE	tinySAK and tinyNET
tinyXCAP (XCAP implementation) Prefix: `txcap`	➢ AUID manager ➢ URL generator	tinySAK, tinyNET and tinyHTTP
tinySMS (Binary SMS over IMS/LTE networks) Prefix: `tsms`	➢ SM-TL (3GPP TS 23.040) ➢ SM-RL (3GPP TS 24.011) ➢ SMS over 3GPP IMS/LTE networks (3GPP TS 24.341)	tinySAK
tinySigComp (Signalling Compression) Prefix: `tcomp`	➢ Stream and Dgram compression/decompression ➢ SIP/SDP and Presence dictionaries ➢ Deflate Compressor ➢ UDVM	tinySAK
tinyIPSec (IPSec) Prefix: `tipsec`	➢ Mode: Transport and Tunnel ➢ Protocol: AH, ESP or both ➢ IP Protocol: UDP and TCP ➢ Algorithm: HMAC-MD5-96 and HMAC-SHA-1-96 ➢ Encryption Algorithm: NULL, DES-EDE3-CBC and AES	tinySAK
tinyMSRP (MSRP) Prefix: `tmsrp`	➢ Large Message IM (RFC 4975 and OMA SIMPLE IM) ➢ File Transfer (RFC 5547) ➢ Image Sharing (GSMA IR.79)	…
tinySDP (SDP) Prefix: `tsdp`	➢ SDP Offer Answer (RFC 3262 and draft-ietf-sipping-sip-offeranswer-12)	tinySAK
tinyMEDIA (Audio, Video, File Transfer …) Prefix: `tmedia`	➢ Plugins ➢ Codecs	tinySAK, tinyNET and tinySDP
tinyGST (GStreamer) Prefix: `tgst`		
tinySIP (3GPP IMS/LTE framework) Prefix: `tsip`	➢ SIP (RFC 3261, 3GPP TS 24.229 Rel-9) ➢ IMS-AKA (RFC 3310, 3GPP TS 33.203) ➢ IPv4/IPv6 dual stack ➢ UDP, TCP, TLS and SCTP transports ➢ Service-Route Discovery (RFC 3608) ➢ Proxy-CSCF discovery using DHCPv4/v6 or/and DNS NAPTR+SRV ➢ SigComp (RFC 3320, 3485, 4077, 4464, 4465, 4896, 5049, 5112 and 1951)	tinySAK, tinyNET, tinySDP, tinyMEDIA, tinyHTTP and tinyIPSec

	➢ IPSec ➢ Security Agreement (RFC 3329) ➢ NAT Traversal (STUN, TURN and ICE) ➢ Preconditions (RFC 3312, 4032 and 5027) ➢ SMS over IP (3GPP TS 23.038, 24.040, 24.011, 24.341 and 24.451) ➢ ENUM (RFC 3761) ➢ The tel URI for Telephone Numbers (RFC 3966) ➢ SIP SIMPLE (Presence subsciption/publication, Pager Mode IM, ...) ➢ MMTel (UNI) ➢ SDP Offer-Answer (SOA) ➢ Session Timers ➢ File transfer (RFC 5547) and Image Sharing(GSMA IR.79) ➢ Large Message IM (OMA SIMPLE IM) ➢ To be continued....	

23.1 GNU Build Tools

23.1.1 Cross-compilation

23.2 Windows XP/Vista/7

The source code contains both Visual Studio 2005 and 2008 projects. Visual Studio 2005 project (*.vcproj) and solution (*.sln) files are under `$(DOUBANGO_HOME)/vs_2005/tiny*`. For VS2008 projects, these files are on the root directory of each project. Both Visual Studio Professional and Express editions could be used. You only need the C/C++ editions.

As all other systems you can use GNU Build Tools with MinGW or Cygwin. For more information, please refer to section 23.1.

23.3 Windows Mobile

You can use the Visual Studio Projects described above. You should install "Windows Mobile SDK 5.0 for Pocket PC" or later.

Cross-compilation with MinGW or Cygwin could be achieved by using CeGCC. For more information, please refer to subsection 23.1.1.

23.4 MAC OS X/iPhone/iPad

The source code contains Xcode project files under `$(DOUBANGO_HOME)/xcode/tiny*`.

As stated above, GNU build tools use `libtool` which used `libtoolize` command to generate shared libraries. Mac OS X does ship with the `libtoolize` command, but due to conflict, it has been renamed to `glibtoolize` (for GNU libtoolize). In order for the `autogen.sh` scripts to be launched, you have to specify the environment variable `LIBTOOLIZE` as follow:

```
$> LIBTOOLIZE="glibtoolize" ./autogen.sh
```

Note for Mac OS X 10.6 (Snow Leopard): Snow Leopard is an hybrid operating system that can run on both 32 and 64 bits applications, and even mix the two addressing mode. By default, the GCC compiler targets the x86_64 architecture, even on 32 bits kernels. Therefore, you need to explicitly set the architecture to build when using the auto-tools:

For i386 architecture:

```
$> CLFAGS="-arch i386" LDFLAGS="-arch i386" ./configure
```

For x86_64 architecture:

```
$> CLFAGS="-arch x86_64" LDFLAGS="-arch x86_64" ./configure
```

23.5 Unix-Like systems

23.6 µLinux

23.7 Google Android

23.7.1 Shared Libraries

Because there is an issue with GNU libtool ARM cross-compilation (only static libraries), you should use the Android special makefiles (under `$(DOUBANGO_HOME/android-projects`)) in order to build shared libraries. Shared libraries are required for those who want to use doubango in their Java (JNI/Dalvik) or Mono (P/Invoke) programs.

1) Adjust `$(DOUBANGO_HOME)/android-projects/root.mk`:
   a) Change `$ANDROID_NDK_ROOT` variable to point to the NDK root directory (e.g. `/cygdrive/c/android-ndk`)
   b) Change `$ANDROID_SDK_ROOT` variable to point to the SDK root directory (e.g. `/cygdrive/c/android-sdk`)
   c) Change `$ANDROID_PLATFORM` variable to point to your preferred platform root directory (e.g. `$(ANDROID_NDK_ROOT)/build/platforms/android-1.5`)
   d) To change the default directory where libraries are deployed, change `$LIB_DIR` variable. Because the dynamic linker does not honor the `LD_LIBRARY_PATH`, this variable should always point to `/system/lib`.
2) Open new Console window
3) Add SDK TOOLS root directory to the system `$PATH` if not already done:
   a) `export PATH=$ANDROID_SDK_ROOT:$PATH`
4) Add the toolchain root binary directory to the system PATH if not already done:
   a) `export PATH=$ANDROID_NDK_ROOT\build\prebuilt\$(HOST)\arm-eabi-4.2.1\bin:$PATH`

where `$HOST` is equal to `darwin-x86` on MAC OS X, `windows` on Windows XP/Vista/7 and `linux-x86` on Unix-like systems.

5) Set your custom `$(CFLAGS)` flags to change the default behavior:
   a) Example: `export CFLAGS="-g3 -O0 -DDEBUG_LEVEL=DEBUG_LEVEL_INFO"`. You can off course set any valid GCC `$(CFLAGS)` flags.
6) Go to the android-projects root directory:
   a) `cd $(DOUBANGO_HOME)/android-projects`
7) Build and deploy a project:
   a) `make PROJECT=tiny* all`

   where `tiny*` is equal to `tinySAK` or `tinyNET` or `tinyHTTP` or …. For example, to build `tinySAK` project, type `make PROJECT=tinySAK all`. This will build and deploy the project on the device or emulator. All libraries will be deployed to `$LIB_DIR` (default value=`/system/lib`) and executable to `$EXEC_DIR` (default value=`/data/tmp`).

23.8 Symbian S60

The easiest way to compile, test and deploy your application on S60 systems is to use Carbide projects which are under `$(DOUBANGO_HOME)/carbide/tiny*`.

24 Demonstration

To show how powerful the 3GPP IMS/LTE stack is, we have developed a tiny but fully featured demo program. The demo program is name tinyDEMO and is part of the source code.

tinyDEMO is a command-line program working on any system supported by doubango framework (such as Android, Symbian S60, Windows Mobile, iPhone, µLinux, Windows XP/Vista/7, MAC OS X, Unix-Like systems …). It is used to test all SIP and media features. The program offers an elegant and easy way to execute many commands by using scenarios. For more information about scenarios, please refer to the section 24.2.

24.1 DSSL

DSSL stands for **D**oubango **S**imple **S**cripting **L**anguage and is used to write scenarios. For more information, please refer to the next sections.

24.1.1 Synopsis

Usage: **`++command (--[option] @@[level])`**

`command` is mandatory and should start the command-line. You can write a command on several lines by using back-slashes (`\`). Both `option` and `level` are optional.

24.1.2 Options

Options are used to configure a command.

Option	Alias	Description	Example
amf [value]			--amf 0x8FF
caps [value]			--caps +g.oma.sip-im or --caps language=en,fr
dhcpv4			--dhcpv4
dhcp6			--dhcpv6
dname [value]		Display name	--dname bob
dns-naptr			--dns-naptr
from [value]		Sets the Source URI. [value] should be a valid SIP/SIPS/TEL URI.	--from sip:bob@open-ims.test
expires [value]	xp	[value] is in seconds.	--expires 30
header [name=value]			--header Supported=norefersub
impi [value]			--impi bob@open-ims.test
impu [value]		[value] should be a valid SIP URI.	--impu sip:bob@open-ims.test
ipv6			--ipv6
local-ip [value]			--local-ip 192.168.16.104
local-port [value]		[value] should be an unsigned 16-bit value.	--local-port 5060
opid [value]			--opid 0xA712F5D04B
password [value]	pwd		--pwd mysecret
path [value]			--path /c/doubango/sample.sn
payload [value]	pay		--pay hello world
pcscf-ip [value]			--pcscf-ip fe00::
pcscf-port [value]			--pcscf-port 4060
pcscf-trans [value]		[value] should be: udp or tcp or tls or sctp.	--pcscf-trans tcp
realm [value]			--realm open-ims.test
sec [value]			--sec 3600
sid [value]			--sid 123
sigcomp			--sigcomp
smsc [value]		[value] should be a valid SIP/SIPS/TEL URI. The userinfo part should be a valid E.164 number.	--smsc sip:+3315245856@open-ims.test;user=phone
to [value]		[value] should be a SIP or tel URI.	--to sip:alice@open-ims.test

24.1.3 Levels

Levels are optionally used to set the session scope. An alias is a short name used to identify an option.

Level	Alias	Description	Example
stack	st	Stack-level option	@@st
session	ss	Session-level option	@@ss
action	request, a, r	Action-level option	@@a

24.1.4 Commands

An alias is a short name used to identify a command.

Command	Alias	Required Options	Optional Options	Description
audio[2]	a		to	Makes audio call.
audiovideo[2]	av		to	Makes audio/video call.
config-session	css	sid[1]		Configures any session (REGISTER, PUBLISH, INVITE, SUBSCRIBE …).
config-stack	cst		header, impi, impu, realm, local-ip, local-port	Configures the IMS/LTE stack.
dump	d			Prints all sessions with their ids.
exit	exit, e, quit, q			Quits the application. Before exiting all sessions will be automatically hung up.
file	f	path	to	Send a file using MSRP protocol.
hangup[2]	hu	sid[1]		Hang up (unregister, unsubscribe, bye …) a session.
help	h			Shows this help.
large-message[2][3]	lm	pay	to	Sends Large message using MSRP (as per OMA SIMPLE IM).
message[2][3]	m	pay	to	Sends a Pager Mode IM.
options[2]	opt		to	Sends OPTIONS request.
publish[2][3]	pub		header, pay, sid[1], to	Sends a PUBLISH request. If --sid is use, then the PUBLISH will reuse this session (refresh). You can use --pay to set the payload.
register[2][3]	reg		header, pay, sid[1], to	Sends a REGISTER message. If --sid is use, then the REGISTER will reuse this session (refresh).
run	r			Runs the stack. Before to try to run the stack you should set the impi, impu and realm parameters which are mandatory. Optionally, you can set the IP address and Port of the Proxy-CSCF. All these parameters are defined by using ++cst command. This command is mandatory before starting to send or receive any SIP message.
scenario	sn	path		Load a scenario from a file.
sleep		sec		Pauses the current thread. Use --sec to define the number of seconds to wait. Only the current (UI) thread will be pause. The stack will continue to run. If the number of seconds is <= 0, then the thread will pause until the user press ENTER.
sms[2][3]		pay	to	Sends Binary SMS (RP-DATA). The payload should be in ascii string. It's up to the stack to convert it to GSM 7-bit alphabet.
stop				Stop the stack without exiting the application.
subscribe[2][3]	sub		header, pay, sid[1], to	Send SUBSCRIBE message. If --sid is use, then the SUBSCRIBE will reuse this session (refresh). You can use --pay to set the payload.
video[2][3]	v		header, pay, to	Makes video call.

[1] Use ++dump to retrieve the sid (a.k.a session identifier)

[2] This command should only be executed if the stack is running. To run the stack, use ++run

[3] `to` option is not required but should be set, otherwise, the request will be sent to yourself

Examples:

Configure the Stack (`++cst` or `++config-stack`)

```
++cst --realm doubango.org --impi bob@open-ims.test --impu sip:bob@open-ims.test --pwd
mamadou --header Privacy=header;id --header Allow=INVITE, ACK, CANCEL, BYE, MESSAGE,
OPTIONS, NOTIFY, PRACK, UPDATE, REFER --header P-Access-Network-Info=ADSL;utran-cell-
id-3gpp=00000000 --header User-Agent=IM-client/OMA1.0 doubango/v1.0.0
```

This command will configure the stack by setting the realm, impi, impu, password and four SIP headers. These headers will be added at stack-level.

Send SUBSCRIBE (`++sub` or `++subscribe`):

```
++sub --to sip:mamadou@ims.inexbee.com --expires 3600 --header Event=reg --header
Accept=application/reginfo+xml --header Allow-Events=refer, presence, presence.winfo,
xcap-diff, conference --header MyHeader1=Value1 @@action
```

This command will send a SIP SUBSCRIBE message. Event, Allow-Events and Accept headers will be added as session-level headers as the level option is missing. However, MyHeader1 will be added as action-level (a.k.a request-level) header. Please refer to the previous chapters to understand how levels are used.

For example, the command below shows how to hang-up the SUBSCRIBE session assuming that its id is equal to 2.

```
++hu --sid 2
```

Sleep 2.5 seconds:

```
 ++sleep --sec 2.5
```

24.2 Scenarios

Scenarios are configuration files written using dssl language and used to automate your tests. A scenario is interpreted line by line and should contain less than 4096 bytes (before to be pre-processed). To overcome this limitation, you should use embedded scenarios as explained below.

In the new releases, the scenarios will support more features (filtering, loops, conditions, jump …) to ease testing complex use-cases. This is why you should adopt this tool if you want to be able to easily test advanced features (MMTel, PoC …).

For newbies, a sample scenario (`sample.sn`) can be found on tinyDEMO root directory.

24.2.1 Synopsis

At start-up, or at any time, you can load and execute a scenario like this: **`demo.exe ++sn ./sample.sn`**. You can continue to type in your commands to the ‘command prompts’ in parallel with the execution of the scenario.

To explain how scenarios could be written, we will analyse an example file (reproduced below) step by step. This file contains 14 steps (from 0-13).

```
## For more information about scenarios, please refer to the Programmer's Guide


# 0 - user's parameters (like ANSI-C #define)
%%domain open-ims.test
%%user bob
%%pwd mysecret
%%proxy_ip 192.168.0.13 # IP address or FQDN
%%proxy_port 4060
%%proxy_trans udp # udp, tcp, tls or sctp
%%expires 100 # expires used by all dialogs
%%sleep-sec 1.0 # number of seconds to wait before sending next sip message


%%reg-sid
%%sub-reg-sid
%%sub-pres-sid
%%pub-sid
```

```
# 1 - configure the stack
++cst --realm $$(domain) --impi $$(user)@$$(domain) --impu sip:$$(user)@$$(domain) --pwd $$(pwd) \
      --pcscf-ip $$(proxy_ip) --pcscf-port $$(proxy_port) --pcscf-trans $$(proxy_trans)\
      --header Privacy=header;id --header Allow=INVITE, ACK, CANCEL, BYE, MESSAGE, OPTIONS, NOTIFY, PRACK, UPDATE, REFER \
      --header P-Access-Network-Info=ADSL;utran-cell-id-3gpp=00000000 \
      --header User-Agent=IM-client/OMA1.0 doubango/v1.0.0 # last should not have backslash

# 2 - Run the engine
++r

# 3 - REGISTER
++reg --xp $$(expires) --caps +g.oma.sip-im --caps +g.3gpp.smsip --caps language="en,fr" --header Action-Header=Myheader-value @@action >>(reg-sid)

# 4 - sleep
++sleep --sec $$(sleep-sec)

# 5 - subscribe to reg event package
++sub --to sip:$$(user)@$$(domain) --xp $$(expires) --header Event=reg --header Accept=application/reginfo+xml \
      --header Allow-Events=refer, presence, presence.winfo, xcap-diff, conference >>(sub-reg-sid)

# 6 - subscribe to presence event package (alice)
++sub --to sip:alice@$$(domain) --xp $$(expires) --header Event=presence --header Supported=eventlist\
      --header Accept=multipart/related, application/rlmi+xml, application/pidf+xml, application/rpid+xml, application/xcap-diff+xml, message/external-body \
      --header Allow-Events=refer, presence, presence.winfo, xcap-diff, conference >>(sub-pres-sid)

# 7 - publish (like ANSI-C #include)
++sn --path ./publish.sn # embedded scenario

# 8 - sleep until the user press ENTER
++sleep --sec -1 # zero also work

# 9 - unsunscribe (reg and pres) and unpublish
++hu --sid $$(sub-reg-sid)
++hu --sid $$(sub-pres-sid)
++hu --sid $$(pub-sid)

# 10 - sleep
++sleep --sec $$(sleep-sec)
```

```

# 11 - unregister
++hu --sid $$(reg-sid)


# 12 - sleep
++sleep --sec $$(sleep-sec)


# 13 - Exit the application
++e
```

example.sn

All commands (**++**), options (**--**) and levels (**@@**) used in this file are explained in the previous sections. There is the list of all reserved symbols:

1. **#** is used to comment (ignore) lines. This symbol can be used at the beginning or middle of a line.
2. **++** is used to execute a command. Should always be at the beginning of a line. Supported commands are listed at section 24.1.4.
3. **--** is used to set command options. Should always be used with a command. Supported options are listed at section 24.1.2.
4. **@@** is used to defined the level at which the option should be used. Supported levels are listed at section 24.1.3.
5. **%%** is used to declare user's parameters. It is equivalent to ANSI-C #define macro. In our example, `%%domain open-ims.test` is equivalent to `#define domain "open-ims.test"`.
6. **$$** is used to print the value of a previously defined user's parameter. '(' and ')' parentheses are optionally used to delimit the name of the parameter. In our example, `$$(domain)` will print `open-ims.test`.
7. **>>** is used to redirect the sid (session id) value into a user's parameter. This parameter could be used later to hang-up or refresh a session.
8. \ (backslash) is used to write multi-line commands. Each line should ends with this symbol, except the last one.

Step-0: Declare all user's parameters. `reg-sid`, `sub-reg-sid`, `sub-pres-sid` and `pub-sid` are valueless parameters used to store dynamically generated session ids (sid). User's parameters are cross-scenario, which means that embedded scenarios (++sn) can also reference them. An example of embedded scenario is given at **step-7**. This embedded scenario (publish.sn) will use `pub-sid` to store the value of its session identifier. As a scenario is executed line by line, you should declare (**%%**) your parameters before beginning to reference (**$$**) them.

Step-1: Configures the stack. `++cst` is short for `++config-stack`. For example, `--impu sip:$$(user)@$$(domain)` is used to set the user's IMPU (IMS Public Identity). The preprocessor will change this former statement into `--impu sip:bob@open-ims.test`. As there is no level, all headers (--header) will be added at stack level.

Step-2: Runs the stack. `++r` is short for `++run`. This command is mandatory and should be executed before starting to send any SIP message.

Step-3: Sends SIP REGISTER message. `++reg` is short for `++register`. And `++xp` is short for `++expires`. `--caps` option is used to set the user's capabilities. The session id, which is automatically generated (for uniqueness) by the stack, is stored (>>) in `%%reg-sid` parameter for later use. The header added to the request will be referenced at action-level (@@), which means that only the initial REGISTER will contain this header. If the level option was missing, then this header would be added at session-level (both initial and subsequent requests).

Step-4: Pauses (1.0 second) the user's main thread. The stack will continue to run in background. In this case, we use this command to give the stack the time to complete the registration process. In next releases, condwaits will be used.

Step-5: SUBSCRIBE to reg event package. `++sub` is short for `++subcribe`. The session id will be stored (>>) in `%%sub-reg-sid` parameter for later use. As there are no levels, all options will be added at session-level. In this command, `--to sip:$$(user)@$$(domain)` option is used to set the destination SIP URI (To header and Request URI). The preprocessor will change this former statement into `--to sip:bob@open-ims.test`.

Step-6: SUBSCRIBE to presence event package. See **Step-5** for more description.

Step-7: Execute an embedded scenario. At this step, the interpreter will read, load, preprocess then execute this external scenario. This means that the commands defined in this external scenario will be executed before steps 8-13. You can use this command (`++sn`) into scenarios to form an execution chain (like ANSI-C `#include`). All parameters defined in one scenario will be visible to all others, unless they are declared after the scenario is loaded.

Step-8: See **Step-4** for more information. The difference with **step-4** is that the main thread will pause until the user presses ENTER as the number of seconds is <=0 (-1 in this case).

Step-9: Sends unsubscribe requests for both presence and reg event packages. `++hu` is short for `++hangup`. `--sid` option is used to define which session to hang-up.

Step-10: See **Step-4** for more information.

Step-11: Sends unregister request. `++hu` is short for `++hangup`. `--sid` option is used to define which session to hang-up.

Step-12: See **Step-4** for more information.

Step-13: Exits the application: `++e` is short for `++exit`. When you call this command before hanging up all sessions, then the stack will properly hang them up by starting to shutdown non-register sessions.

24.2.2 Hacking the AOR

On some emulators, such as Google Android or Windows mobile, the IP address and port on which we bound to is not routable. This IP address and port is used in the "Contact" header to form our AOR. On Windows Mobile, the workaround is to cradle the emulator in order to have the same address than the host machine. On Google Android, you should set the value of `HACK_AOR` macro (from `$(DOUBANGO_HOME)/tinyDEMO/demo_config.h`) to 1 and change your scenario to send a SIP OPTIONS (`++opt`) before any other request. This only works if your server supports **rport** as per RFC 3581 and OPTIONS requests as per RFC 3261 section 11. `++sleep` command could be used to allow the OPTIONS request<->response to complete before sending the next requests.

24.2.3 Common Use-Cases

For all use-cases listed below, you can use backslash (\) characters to write your command on several lines.

1. Publish your presence

```
++pub --to sip:$$(user)@$$(domain) --header Content-Type=application/pidf+xml @@a --pay
```

```
<?xml version=1.0 encoding=utf-8?> … @@a
```

2. Subscribe to Alice’s presence

```
++sub --to sip:alice@$$(domain) --xp $$(expires) --header Event=presence --header Sup-
ported=eventlist --header Accept=multipart/related, application/rlmi+xml,
application/pidf+xml, application/rpid+xml, application/xcap-diff+xml, message/extern-
al-body --header Allow-Events=refer, presence, presence.winfo, xcap-diff, conference
>>(sub-pres-sid)
```

3. Sending Pager Mode IM to Alice

```
++m --to sip:alice@$$(domain) --header NS=imdn <urn:ietf:params:imdn> --header imd-
n.Message-ID=34jk324j --header DateTime=2006-04-04T12:16:49-05:00 --header imdn.Dispos-
ition-Notification=positive-delivery, negative-delivery --header Content-
Type=text/plain --pay Hello World
```

4. Sending Binary SMS

```
++sms --smsc sip:+33185145269@$$(domain);user=phone;npi=ISDN --to sip:+33685214585@$$
(domain);user=phone --header Content-Type=application/vnd.3gpp.sms --header Transfer-
-Encoding=binary --pay hello world
```

The SMS will be sent as per 3GPP TS 24.341. The SMSC address will be used as the Request URI (and to populate “To” header) of the SIP MESSAGE. The destination telephone number (--to) will be binary encoded into the RP-DATA message.

If the SMSC address (or destination URI) is a tel URI (E.164), then you should use ENUM protocol to obtain a valid SIP URI. ENUM is supported by tinyNET and a short example is given in section 11.2 ENUM.

For Mobile Terminated RP-DATA message carrying SMS-DELIVER (RP-UD) payload, the demo program will automatically send an RP-ACK (SMS-DELIVER-REPORT) message. RP-ACK (SMS-SUBMIT-REPORT) messages received from the IP-SM-GW will be parsed and printed to the console to show how reports are handled.

5. Sending Large Message (MSRP)
6. Sending a file (MSRP)
7. Making Audio Call
8. Making Video Call
9. Making Audio/Video Call
10. Making Audio/Video Conference