How-To initialize libeXosip2.

The eXtented eXosip stack

Initialize eXosip and prepare transport layer

When using eXosip, your first task is to initialize both eXosip context and libosip library (parser and state machines). This must be done prior to any use of libeXosip2.

Also, you need to prepare the transport layer and choose either UDP, TCP, TLS or DTLS.

Here is the basic mandatory setup:

- Initialize the osip trace (compile this code with -DENABLE_TRACE)

```
#include <eXosip2/eXosip.h>

eXosip_t *ctx;
int i;
int port=5060;

TRACE_INITIALIZE (6, NULL);
```

- Initialize eXosip (and osip) stack

```
ctx = eXosip_malloc();
if (ctx==NULL)
  return -1;

i=eXosip_init(ctx);
if (i!=0)
  return -1;
```

- Open a UDP socket for signalling

```
i = eXosip_listen_addr (ctx, IPPROTO_UDP, NULL, port, AF_INET, 0);
if (i!=0)
  {
    eXosip_quit(ctx);
    fprintf (stderr, "could not initialize transport layer\n");
    return -1;
  }
```

Choosing UDP, TCP, TLS or DTLS

If you wish to use other transport protocol, you can select:

- **UDP:**

```
i = eXosip_listen_addr (ctx, IPPROTO_UDP, NULL, 5060, AF_INET, 0);
```

- **TCP:**

```
i = eXosip_listen_addr (ctx, IPPROTO_TCP, NULL, 5060, AF_INET, 0);
```

- **TLS:**

```
i = eXosip_listen_addr (ctx, IPPROTO_TCP, NULL, 5061, AF_INET, 1);
```

- **DTLS:**

```
i = eXosip_listen_addr (ctx, IPPROTO_UDP, NULL, 5061, AF_INET, 1);
```

Specific setup for TLS

TLS may requires specific setup. TLS introduce in fact two interesting features:

- Using certiticates and keys, it helps to trust/verify the remote server.
- It also encrypts data so that no man-in-the-middle could read the SIP traffic.

If you don't need server verification, TLS is very easy to setup. You don't need to configure any certificate, key or root certificate...

Here is the code to disable certificate verification:

```
int val=0;
i = eXosip_set_option (ctx, EXOSIP_OPT_SET_TLS_VERIFY_CERTIFICATE, (void*)&val);
```

If you require validation, a few work still needs to be done. What you need depends on your platform/os.

- Certificate on windows:

On windows, exosip has built-in support for "Windows Certificate Store". Thus, you only need to add your certifcate and keys in the official "Windows Certificate Store".

- Certificate on macosx:

On macosx, exosip has built-in support for the certificate store.

- Certificate on other platforms:

```
eXosip_tls_ctx_t tls_info;
memset(&tls_info, 0, sizeof(eXosip_tls_ctx_t));
snprintf(tls_info.client.cert, sizeof(tls_info.client.cert), "user-cert.crt");
snprintf(tls_info.client.priv_key, sizeof(tls_info.client.priv_key), "user-privkey.cr
snprintf(tls_info.client.priv_key_pw, sizeof(tls_info.client.priv_key_pw), "password"
snprintf(tls_info.root_ca_cert, sizeof(tls_info.root_ca_cert), "cacert.crt");

i = eXosip_set_option (ctx, EXOSIP_OPT_SET_TLS_CERTIFICATES_INFO, (void*)&tls_info);
```

Additionnal setup

A few options can be modified in eXosip2. However, most default are good values and if your sip service is configured correctly, no much settings beyond default would be required.

- EXOSIP_OPT_UDP_KEEP_ALIVE 1
- EXOSIP_OPT_UDP_LEARN_PORT 2
- EXOSIP_OPT_USE_RPORT 7
- EXOSIP_OPT_SET_IPV4_FOR_GATEWAY 8
- EXOSIP_OPT_ADD_DNS_CACHE 9
- EXOSIP_OPT_DELETE_DNS_CACHE 10
- EXOSIP_OPT_SET_IPV6_FOR_GATEWAY 12
- EXOSIP_OPT_ADD_ACCOUNT_INFO 13
- EXOSIP_OPT_DNS_CAPABILITIES 14
- EXOSIP_OPT_SET_DSCP 15
- EXOSIP_OPT_REGISTER_WITH_DATE 16
- EXOSIP_OPT_SET_HEADER_USER_AGENT 17

- EXOSIP_OPT_SET_TLS_VERIFY_CERTIFICATE 500
- EXOSIP_OPT_SET_TLS_CERTIFICATES_INFO 501
- EXOSIP_OPT_SET_TLS_CLIENT_CERTIFICATE_NAME 502
- EXOSIP_OPT_SET_TLS_SERVER_CERTIFICATE_NAME 503

Here is a basic setup that might be appropriate for usual configuration:

```
int val;

eXosip_set_user_agent (ctx, "exosipdemo/0.0.0");

val=17000;
eXosip_set_option (ctx, EXOSIP_OPT_UDP_KEEP_ALIVE, (void*)&val);

val=2;
eXosip_set_option (ctx, EXOSIP_OPT_DNS_CAPABILITIES, (void*)&val);

val=1;
eXosip_set_option (ctx, EXOSIP_OPT_USE_RPORT, (void*)&val);

val=26;
eXosip_set_option (ctx, EXOSIP_OPT_SET_DSCP, (void*)&dscp_value);

eXosip_set_option (ctx, EXOSIP_OPT_SET_IPV4_FOR_GATEWAY, "sip.antisip.com");
```

NAT and Contact header

There would be much to say about this. Most being unrelated to the eXosip2 stack itself...

The most important feature with SIP is to be able to receive SIP requests. You wouldn't be glad if your phone remains silent. However, it's in theory not possible to guess what should contain the Contact headers we are creating.

Most proxy will repair our broken contact, no matter why eXosip2 or any SIP application has provided a wrong value. The SIP specification is not very clear on the various client and server behaviors for Contact verifications.

However:

1. No matter, what eXosip2, most proxy will repair correctly.
2. No matter what you think is right, some people think another way is right.
3. No matter sip, proxy and people: **NETWORK RULES ALWAYS APPLIES FIRST! ;)**

Anyway, to avoid problems:

1. You should always do your best to put correct information in Contact.
2. Most of the time, you can't, but it should work anyway.

Conclusion:

1. Without any configuration (NAT, STUN, etc), your proxy should be able to find out how to reach you (over the existing connection).
2. If it can't (whatever the reason), you can try workarounds and options.

The options you have:

- This option helps exosip2 to detect the localip when you have several ones: (VPN and eth0 for example). This will helps to detect localip for Via and Contact. The usual parameter is the proxy. (cautious: *method may block because of DNS operation*)

  ```
  eXosip_set_option (ctx, EXOSIP_OPT_SET_IPV4_FOR_GATEWAY, "sip.antisip.com");
  ```

- masquerading: When sending your first SIP request (REGISTER? OPTIONS?), the top Via of the answers will contain a "received" and "rport" parameters: those IP/port are the exact ones required for your Contact headers with the proxy. STUN will detect similar IP/port but for another destination (the stun server). Thus, STUN is not the correct way.

  Thus, send a request to your proxy, check the via (received and rport) parameter and use masquerading:

```
eXosip_masquerade_contact (ctx, "91.121.81.212", 10456);
```

- EXOSIP_OPT_UDP_LEARN_PORT option: If you wish to re-use Via "received" and "rport" automatically with UDP. With the following code, the second REGISTER (after authentication?) or second outgoing REQUEST will contains the masqueraded Contact header. It should also be used if you masquerade using STUN values.

```
eXosip_masquerade_contact (ctx, "192.168.2.1", 5080);
val=1;
eXosip_set_option (ctx, EXOSIP_OPT_UDP_LEARN_PORT, &val);
```

- EXOSIP_OPT_USE_RPORT option: *only use with BROKEN nat*. This option remove the "rport" parameter in outgoing REQUEST. This should be never used.

```
val=0;
eXosip_set_option (ctx, EXOSIP_OPT_USE_RPORT, &val);
```

If you still have NAT issue, think about using TLS: broken NAT sometimes block SIP packets, but with encryption, broken NAT can't do anything!

About DNS

eXosip2 should be compiled with c-ares. This is very important as c-ares provides non blocking, portable and full support for all required SIP operation.

By default, SIP requires usage of specific DNS features to discover the IP address of a sip service.

- NAPTR to discover the SRV records
- SRV records are then used to receive the list of hosts.
- DNS resolution provide IPs of the host.

For complete information, check rfc3263.txt: Locating SIP servers.

- To use NAPTR:

```
val=2;
eXosip_set_option (ctx, EXOSIP_OPT_DNS_CAPABILITIES, &val);
```

If NAPTR is not set for the service you used which happens in many case, SRV record or normal DNS will be used as a fallback. It should not slow too much the procedure. However, it is still usefull in some case to disable NAPTR because there still exist a few DNS server that remains silent when sending NAPTR request. In that very specific use-case, this may lead to very slow fallback to normal DNS...

- To not use NAPTR:

```
val=0;
eXosip_set_option (ctx, EXOSIP_OPT_DNS_CAPABILITIES, &val);
```

Handle eXosip2 events (eXosip_event_t)

The **eXosip_event** contains all information you need:

- *rid*: identifier for registrations.
- *tid*: identifier for transactions.
- *cid*, did identifiers for calls.
- *sid*, did: identifier for outgoing subscriptions.
- *nid*, did: identifier for incoming subscriptions.
- *request*: outgoing or incoming request for the event
- *answer*: outgoing or incoming answer for the event
- *ack*: outgoing or incoming ACK for the event

Those identifiers are re-used in related eXosip2 API to make it simpler to control context. The request, answer and ack are fully duplicated so that you can access them without locking the eXosip2 context.

Now you have to handle eXosip events. Here is some code to get **eXosip_event** from the eXosip2 stack.

Note: For advanced users, or more real time app, you may also use lower level API so that you can get woken up on a select call when an event occurs.

```
eXosip_event_t *evt;
for (;;)
  {
    evt = eXosip_event_wait (ctx, 0, 50);
    eXosip_lock(ctx);
    eXosip_automatic_action (ctx);
    eXosip_unlock(ctx);
    if (evt == NULL)
      continue;
    if (evt->type == EXOSIP_CALL_NEW)
      {
      ....
      ....
      }
    else if (evt->type == EXOSIP_CALL_ACK)
      {
      ....
      ....
      }
    else if (evt->type == EXOSIP_CALL_ANSWERED)
      {
      ....
      ....
      }
    else .....
    ....
    ....

    eXosip_event_free(evt);
  }
```

You will receive one event for each SIP message sent. Each event contains the original request of the affected transaction and the last response that triggers the event when available.

You can access all headers from those messages and store them in your own context for other actions or graphic displays.

For example, when you receive a REFER request for a call transfer, you'll typically retreive the "refer-To" header:

```
osip_header_t *referto_head = NULL;
i = osip_message_header_get_byname (evt->sip, "refer-to", 0, &referto_head);
if (referto_head == NULL || referto_head->hvalue == NULL)
```

Here are a few examples:

- Answer 180 Ringing to an incoming INVITE:

```
if (evt->type == EXOSIP_CALL_NEW)
{
  eXosip_lock (ctx);
  eXosip_call_send_answer (ctx, evt->tid, 180, NULL);
  eXosip_unlock (ctx);
}
```

- Answer 200 Ok to an incoming MESSAGE: (also check the attachment in evt->request!)

```
if (evt->type == EXOSIP_MESSAGE_NEW && osip_strcasecmp (minfo.method, "MESSAGE") == 0) {
{
  osip_message_t *answer=NULL;
  int i;
  eXosip_lock (ctx);
  i = eXosip_message_build_answer (ctx, evt->tid, 200, &answer);
  i = eXosip_message_send_answer (ctx, evt->tid, 200, answer);
  eXosip_unlock (ctx);
}
```

- Handle incoming REFER within dialog (call transfer): fg

```
if (evt->type == EXOSIP_CALL_MESSAGE_NEW && osip_strcasecmp (minfo.method, "REFER") == 0) {
  osip_header_t *refer_to = NULL;

  eXosip_lock (ctx);
  i = eXosip_call_build_answer (ctx, evt->tid, 202, &answer);
  i = eXosip_call_send_answer (ctx, evt->tid, 202, answer);

  i = osip_message_header_get_byname (evt->request, "refer-to", 0, &refer_to);
  if (i >= 0) {
    printf ("you must start call to: %s\n", refer_to->hvalue);
    ...
  }
  else {
  }
  eXosip_call_terminate (ctx, evt->cid, evt->did, 486);
  eXosip_unlock (ctx);
}
```