MjSip-Mini-Tutorial

MjSip version: 1.5

Author: Luca Veltri
Date: 24/4/2005
Document version: 0.1

1. Preface

This document describes the structure and use of the MjSip library. The intent is to provide a simple overview of the MjSip stack helping the programmer on developing his/her own SIP-based applications.

In the rest of the document the reader is supposed to be familiar with networking basis and with the SIP signaling protocol [1]. For more information on SIP you can visit the official IETF SIP Working Group pages [2] and/or other resources on the various web sites dedicated on SIP [3] [XX].
MjSip is just one of the several publicly available SIP implementations, some of them are based on JAIN SIP API specification [XX].

2. What MjSip is

MjSip is a compact and powerful SIP library for easily building SIP applications and services. It provides in the same time the SIP APIs and SIP stack implementation bound together in MjSip packages.

SIP (Session Initiation Protocol) is the IETF (Internet Engineering Task Force) signaling standard for managing multimedia session initiation; it is currently defined in RFC 3261 [1].
SIP can be used to initiate voice, video and multimedia sessions, for both interactive applications (e.g. an IP phone call or a videoconference) and not interactive ones (e.g. a Video Streaming), and it is the more promising candidate as call setup signaling for the present day and future IP based telephony services. SIP has been also proposed for session initiation related uses, such as for messaging, gaming, etc.

MjSip includes all classes and methods for creating SIP-based application. It implements the complete layered stack architecture as defined in [1], and is fully compliant with the standard. Moreover it includes higher level interfaces for Call Control.

Specific information on IETF standardization process can be found on the official IETF SIP Working Group site [2].

MjSip is formed by several packages that include:
- standard SIP objects like SIP Messages, Transactions, Dialogs, etc.,
- various SIP extensions already defined within the IETF,
- Call Control APIs,
- a reference implementation of some SIP systems (both servers and UAs).

3. Some reasons to use MjSip

There are several available implementations of SIP, in both Java and C++ programming languages; MjSip is just another one.
The main characteristics of MjSip are:

- it is Java based, so it is cross-platform,
- it is not just a API, but includes the complete SIP stack implementation,
- it is very powerful and is compliant to the IETF RFC 3261 standard and extensions,
- it is simple to use and very simple to extend (well, this is what we think.. ;-)),
- it is very light, and can be use at the same time for both server and light terminal implementations,
- it implements almost the same lower-level interface of JAIN SIP, so resulting almost familiar to programmers that already knows JAIN SIP,
- it adds to this lower-level interface new Call Control APIs very useful for quickly developing complete SIP-based applications,
- it comes with both a stateless and a stateful SIP Server implementation, and a very simple UA.

There are also some feature are still not supported by MjSip, such as:

- TLS transports,
- Compact header field formats (currently they are only partially implemented).

4. MjSip license

MjSip is freely available and can be freely use for research, testing, or non-profit purpose. MjSip is not free for commercial use. If You are intended to ship some commercial products you have to buy a license.
Although the MjSip development is full compliant with the SIP standard, some problems could be still encountered when using with not SIP-compliant software, in case their compatibility with MjSip has still not been tested.

5. A cup of MjSip

[TO DO]…

6. SIP entities

There are different types of SIP systems:

- Terminals: the end-systems that make and receive calls (e.g. soft-phone, ip hard-phones).
- Proxy servers: intermediary systems that act as both a server and a client for the purpose of making requests on behalf of other clients.
- Redirect servers: servers that generate 3xx responses to requests it receives, directing the client to contact an alternate set of URIs
- Registrar servers: Servers that accept REGISTER requests and place the information they receive in those requests into the location service for the domain they handle.

These system may be composed by one or more of the following SIP entities:

- User Agent Client (UAC): A user agent client is a logical entity that creates a new request, and then uses the client transaction state machinery to send it. The role of UAC lasts only for the duration of that transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration of that transaction. If it receives a request later, it assumes the role of a user agent server for the processing of that transaction.
- User Agent Server (UAS): A user agent server is a logical entity that generates a response to a SIP request. The response accepts, rejects, or redirects the request. This role lasts only for the duration of that transaction. In other words, if a piece of software responds to a request, it acts as a UAS for the duration of that transaction. If it generates a request later, it assumes the role of a user agent client for the processing of that transaction.
- User Agent (UA): A logical entity that can act as both a user agent client and user agent server.
- Proxy Agent: The logical entity that implements the functionality proper of a Proxy server.

UAC and UAS may terminate Transactions, Dialogs and Calls (see later). Instead, a stateful Proxy terminates Transactions.

The MjSip library currently comes with a simple and extensible stateless SIP server, that can act as Registrar, Redirect, Outbound-Proxy, or a combination of them (e.g. Proxy+Registrar). The server implements a simple Location Service.

7. MjSip architecture

According to the SIP architecture defined in RFC 3261, the core MjSip is structured in three base layers: Transport, Transaction, and Dialog. On top of these layers, MjSip provides also Call Control and application level layers, with the corresponding APIs.

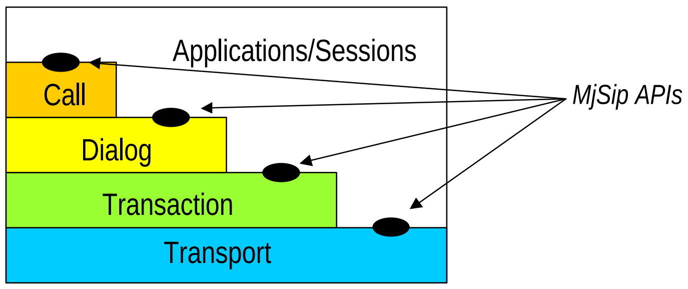


Figure 1 – MjSip layered architecture

7.1. Layered Architecture and APIs

The lowest layer is the *Transport* layer that provides the transport of SIP messages. The **SipProvieder** is the MjSip Object that provides the transport service to all upper layer. It is responsible to send and receive SIP messages through different lower layer transport protocols, and to demultiplex incoming messages toward the appropriate upper layer entity. Every SIP elements should use the SipProvider's API if they want to access to the MjSip transport service.

The second layer is the *Transaction* layer. Transactions are a fundamental component of SIP. In SIP a transaction is a request sent by a client (a transaction client) to a transaction server along with all responses to that request sent from the transaction server back to the client. The transaction layer handles upper-layer retransmissions, matching of responses to requests, and timeouts. The Transaction layer sends and receives messages through the transport layer.

In SIP any task that a user agent client (UAC) accomplishes takes place using a series of transactions. User agents normally should use the transaction layer, as do stateful proxies. Stateless proxies normally do not make use of the transaction layer. As already introduced, the transaction layer has a client component (referred to as a transaction client) and a server component (referred to as a transaction server), each of which are represented by a finite state machine that is constructed to process a particular request. There are defined two kind of transactions:

- *two-way transactions*, and
- *three-way transactions* (currently only the INVITE transaction has been defined, used to initiate a session).

Two-way transactions are implemented in MjSip by **ClientTransaction** and **ServerTransaction**, while three-way/INVITE transactions are implemented by **InviteClientTransaction** and **InviteServerTransaction**.

The third layer (above the transaction layer) is the *Dialog* that binds different transactions within the same “session”. A dialog is a peer-to-peer SIP relationship between two user agents that persists for some time. The dialog facilitates sequencing of messages and proper routing of requests between the user agents. As defined in RFC 2631, the INVITE method establishes a dialog (named invite dialog). An inviete dialog is implemented in MjSip by the class **InviteDialog**. The combination of the *To tag*, *From tag*, and *Call-ID* completely defines a dialog. In the previous RFC 2543, the dialog was called “Call leg”. InviteDialog manages also the CANCEL method.

The upper SIP-layer is the *Call Control* layer that implements a complete SIP call. The Call Control layer is implemented by the **Call** API, that offers a simple-to-use interface to handle incoming and outcoming SIP calls. A Call may consist of more than one dialogs.
Upon the these four layers there are the SIP sessions that bind two or more application entities (participants) on different systems.

MjSip offers APIs for access to all previous SIP layers (from SipProvider to Call), and a reference implementation of various application level systems. A developer can choose to utilize APIs at any level, depending on the developer’s preference and on the system characteristics (perhaps on a transaction-by-transaction basis or direct on the transport service offered by the SipProvider).
The MjSip APIs for the four layers are implemented respectively by:

- class **`Call`** (and class **`ExtendedCall`**)
- class **`InviteDialog`**
- classes **`ClientTransaction`**, **`ServerTransaction`**, **`InviteClientTransaction`**, and **`InviteServerTransaction`**
- class **`SipProvider`**

Moreover, MjSip provides several instruments to handle SIP messages, SDP syntax, and encoding. These methods are included in:

- classes **`Message`**, **`MessageFactory`**
- classes **`Header`**, **`MultipleHeader`**, **`FromHeader`**, **`ToHeader`**, **`ViaHeader`**, and a lot of others..
- classes **`SessionDescription`**, **`Timer`**, etc.

MjSip is intrinsically extensible, and new SIP header and/or new methods can be easily defined and integrated. MjSip stack is formed by two main packages: sip and sipx. The first package includes all SIP standard Headers and Methods, and SIP standard layers (Transport, Transaction, Dialog), while the latter includes possible extensions and it is open for new extensions such as new SIP headers, methods and functions.

In the following sections you can find a brief description of the MjSip main classes.

7.2. SipProvider

The SipProvider sends and receives SIP message. It simply receives SIP messages from an upper layer and sends them to the next-hop SIP entity through the appropriate socket (TCP or UDP). Moreover, it receives SIP messages from the network (through the UDP/TCP layers), and delivers (demultiplexes) them to the appropriate upper entity, that is a Transaction, a Dialog, or an application entity.
When the SipProvider receives a new message from an upper entity, it decides which is the appropriate next-hop SIP agent (addres/port) and transport portocol, and forwards the message to it. In case of SIP requests, the next-hop is chosen depending on the fields To, Route, Contact, and locally configured outbound-proxy, while in case of SIP response it is chosen by the Via sent-by field, received filed, and rport field.
When the SipProvider receives a new message from the lower layers, it chooses the appropriate upper entity based on the matching of transaction-id, dialog-id, or message type with the list of the current registered SipProvider listeners corresponding to the upper entities.

7.3. *Transactions*

The state machines of Non-INVITE Transactions, ,and INVITE Transactions are shown in Figure 2 and Figure 3 respectively.

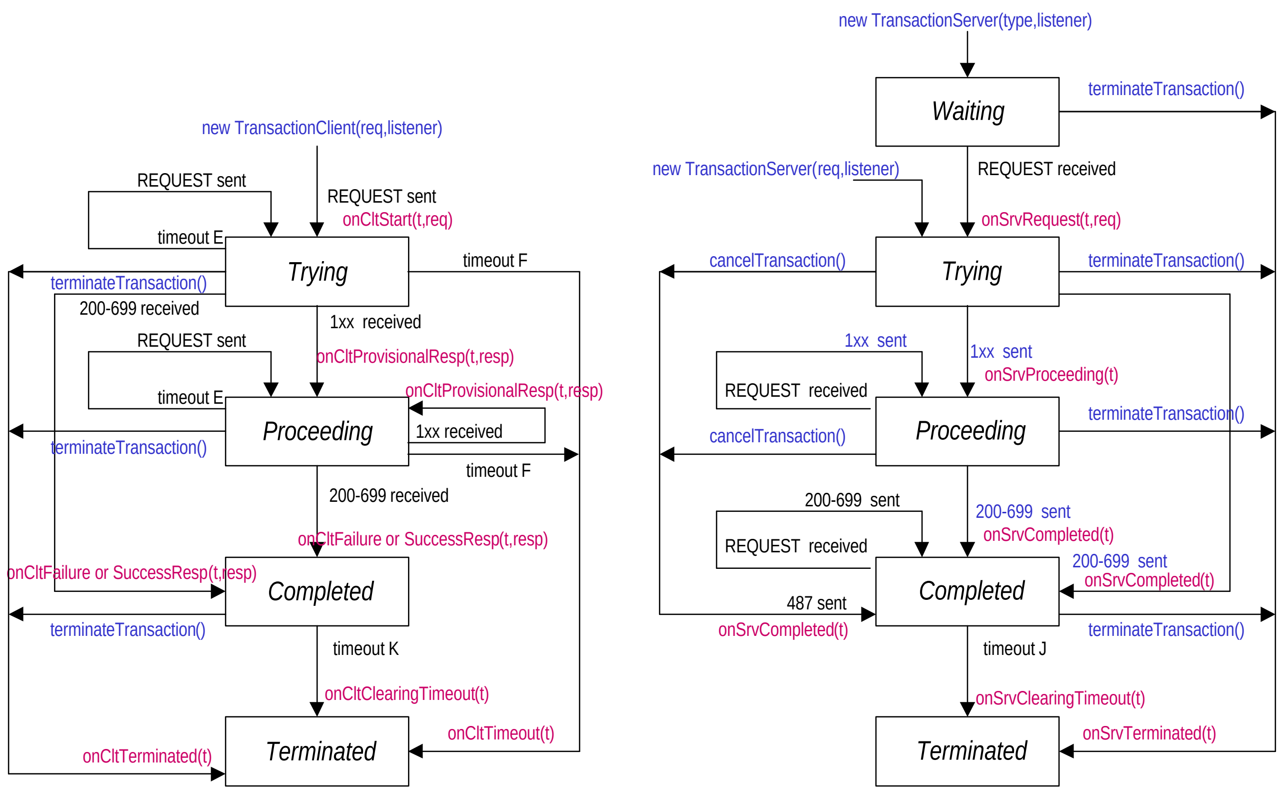


Figure 2 – Non-Invite Transaction Client and Transaction Server.

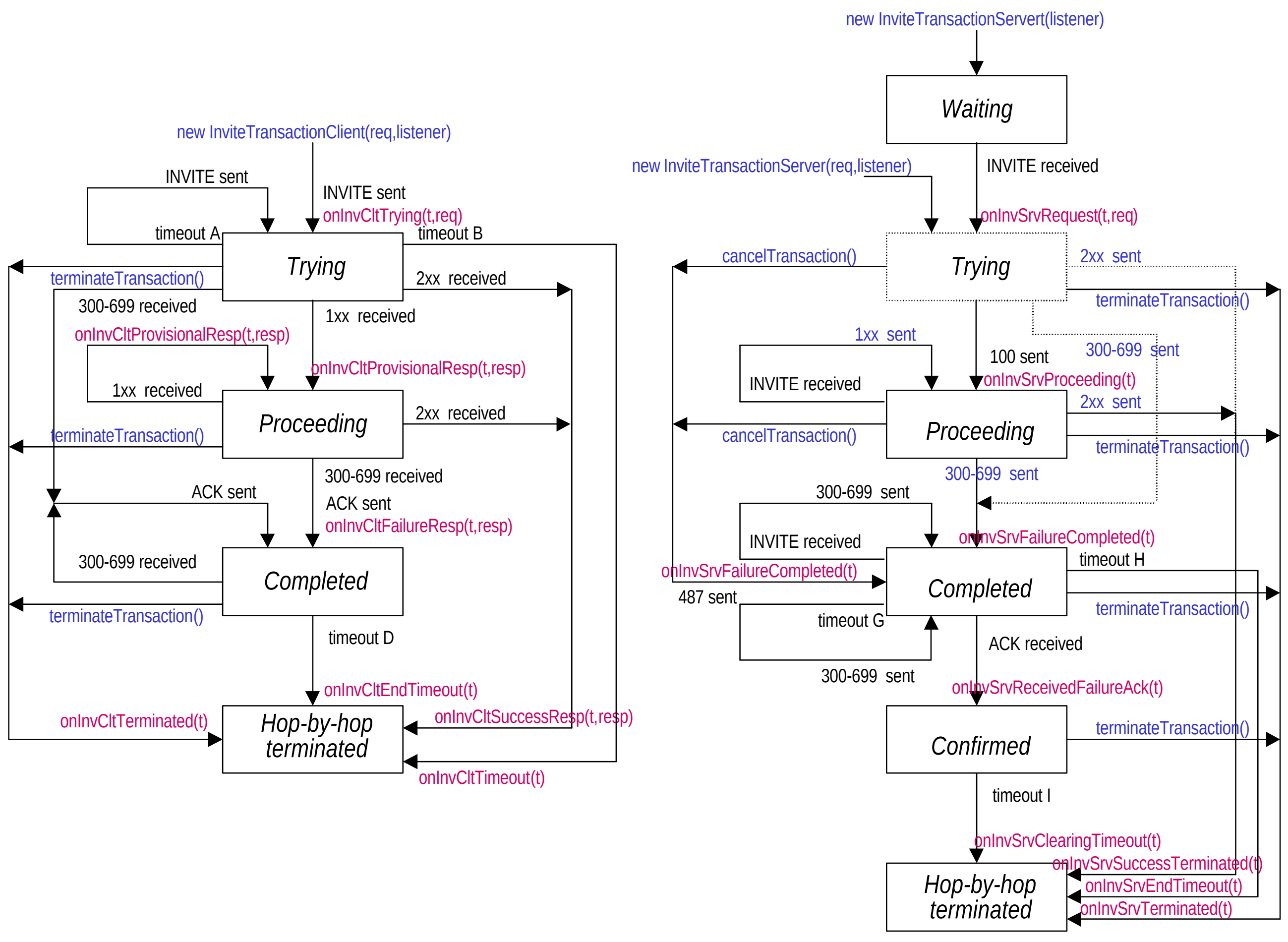


Figure 3 – Invite Transaction Client and Transaction Server.

7.4. Dialogs

Currently only the InviteDialog has been defined (and implemented). The state machine of an InviteDialog is depicted in Figure 4.

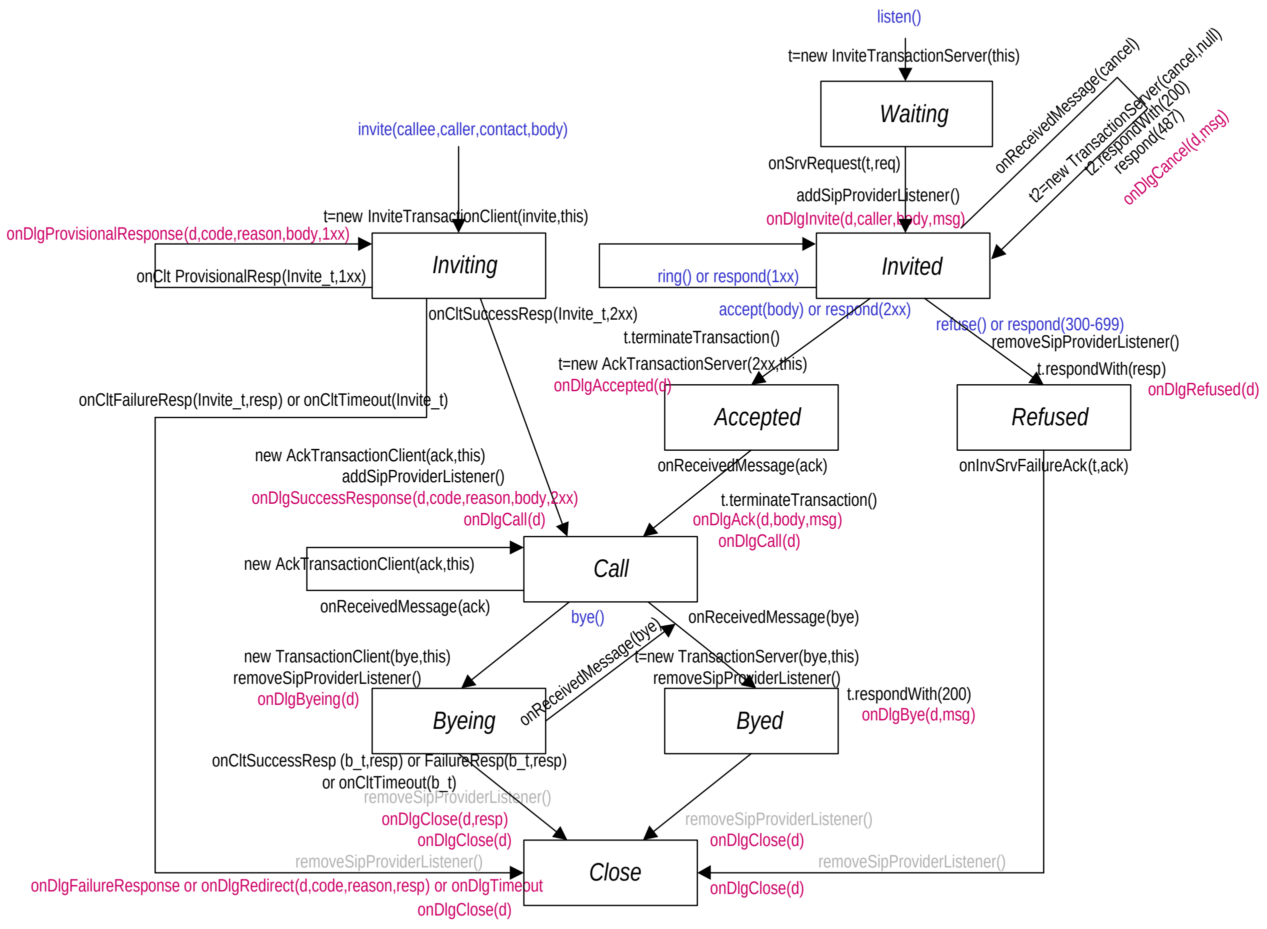

listen()
t=new InviteTransactionServer(this)
Waiting
onSrvRequest(t,req)
addSipProviderListener()
onDlgInvite(d,caller,body,msg)
onReceivedMessage(cancel)
t2=new TransactionServer(cancel,null)
t2.respondWith(200)
respond(487)
onDlgCancel(d,msg)
invite(callee,caller,contact,body)
t=new InviteTransactionClient(invite,this)
onDlgProvisionalResponse(d,code,reason,body,1xx)
onClt ProvisionalResp(Invite_t,1xx)
Inviting
Invited
ring() or respond(1xx)
onCltSuccessResp(Invite_t,2xx)
accept(body) or respond(2xx)
t.terminateTransaction()
t=new AckTransactionServer(2xx,this)
onDlgAccepted(d)
refuse() or respond(300-699)
removeSipProviderListener()
t.respondWith(resp)
onDlgRefused(d)
onCltFailureResp(Invite_t,resp) or onCltTimeout(Invite_t)
Accepted
Refused
onReceivedMessage(ack)
onInvSrvFailureAck(t,ack)
new AckTransactionClient(ack,this)
addSipProviderListener()
onDlgSuccessResponse(d,code,reason,body,2xx)
onDlgCall(d)
t.terminateTransaction()
onDlgAck(d,body,msg)
onDlgCall(d)
new AckTransactionClient(ack,this)
Call
onReceivedMessage(ack)
bye()
onReceivedMessage(bye)
new TransactionClient(bye,this)
removeSipProviderListener()
onDlgByeing(d)
onReceivedMessage(bye)
t=new TransactionServer(bye,this)
removeSipProviderListener()
Byeing
Byed
t.respondWith(200)
onDlgBye(d,msg)
onCltSuccessResp (b_t,resp) or FailureResp(b_t,resp)
or onCltTimeout(b_t)
removeSipProviderListener()
onDlgClose(d,resp)
onDlgClose(d)
removeSipProviderListener()
onDlgClose(d)
removeSipProviderListener()
removeSipProviderListener()
Close
onDlgFailureResponse or onDlgRedirect(d,code,reason,resp) or onDlgTimeout
onDlgClose(d)
onDlgClose(d)

Figure 4 - Invite Dialog.

7.5. Calls

[TO DO]

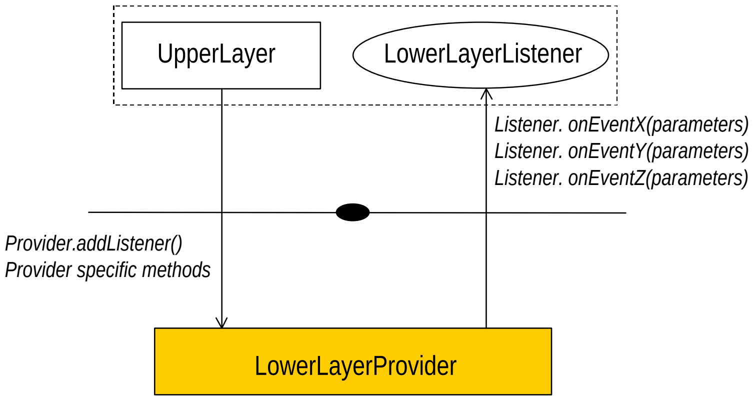
…

8. MjSip layer's APIs

As described in the previous sections, MjSip implements a layered architecture. A developer can choose to use any MjSip layer as he/she prefer. The classes that allow the interaction with the different layers are class **Call**, **InviteDialog**, **ClientTransaction**, **ServerTransaction**, **InviteClientTransaction**, **InviteServerTransaction**, and **SipProvider**.

Class **Call** is the upper API to the service offered by the SIP stack, and offer a interface to the Call layer based on service requests (methods) and service response/indications (callback functions).
Classes **InviteDialog**, **ClientTransaction**, **ServerTransaction**, **InviteTransactionClient**, **InviteTransactionServer**, and **SipProvider** offer interfaces to the lower layers (InviteDialog, Transaction and SipProvider respectively).

The interfaces between adjacent layers are based on a Provider → Listener model. When a class wants to interact with an underling layer, it has to extend the relative LayerListener class for that layer (i.e. the layer provider) and add itself to the list of possible listeners of the events generated by the lower layer/provider. The events are captured by the upper class through specific listener methods inherited by the specific Listener class. Figure 5 shows the API model.


UpperLayer
LowerLayerListener
Listener. onEventX(parameters)
Listener. onEventY(parameters)
Listener. onEventZ(parameters)
Provider.addListener()
Provider specific methods
LowerLayerProvider
Figure 5 – SipDialog, Transaction, SipProvider APIs' model

A SipProviderListener implements only one method, that is the onReceivedMessage(Message msg) method.

A TransactionClient implements the following methods:
- onCltProvisionalResponse(TransactionClient transaction, Message resp), called when the TransactionClient is (or goes) in "Proceeding" state and receives a new 1xx provisional response;
- onCltSuccessResponse(TransactionClient transaction, Message resp), called when the TransactionClient goes into the "Completed" state receiving a 2xx response;
- onCltFailureResponse(TransactionClient transaction, Message resp), called when the TransactionClient goes into the "Completed" state receiving a 300-699 response;
- onCltTimeout(TransactionClient transaction), called when the TransactionClient goes into the "Terminated" state, caused by transaction timeout.

A TransactionServerListener implements the following methods:
- onSrvRequest(TransactionServer transaction, Message req), called when the TransactionServer goes into the "Trying" state receiving a request;
- onInvSrvFailureAck(InviteTransactionServer transaction, Message ack), called when an InviteTransactionServer goes into the "Confirmed" state receining an ACK for NON-2xx response.

A InviteDialogListener implements the following methods:
- onDlgInvite(InviteDialog dialog, NameAddress caller, String body, Message msg), called when an incoming INVITE is received;
- onDlgReInvite(InviteDialog dialog, String body, Message msg), called when an incoming Re-INVITE is received;
- onDlgProvisionalResponse(InviteDialog dialog, int code, String reason, String body, Message msg), called when a 1xx response response is received for an INVITE transaction;
- onDlgSuccessResponse(InviteDialog dialog, int code, String reason, String body, Message msg), called when a 2xx successfull final response is received for an INVITE transaction;

- onDlgRedirectResponse(InviteDialog dialog, int code, String reason, MultipleHeader contacts, Message msg), called when a 3xx redirection response is received for an INVITE transaction;
- onDlgFailureResponse(InviteDialog dialog, int code, String reason, Message msg), called when a 400-699 failure response is received for an INVITE transaction;
- onDlgTimeout(InviteDialog dialog), called when INVITE transaction expires;
- onDlgReInviteProvisionalResponse(InviteDialog dialog, int code, String reason, String body, Message msg), called when a 1xx response response is received for a Re-INVITE transaction;
- public void onDlgReInviteSuccessResponse(InviteDialog dialog, int code, String reason, String body, Message msg), called when a 2xx successfull final response is received for a Re-INVITE transaction;
- onDlgReInviteFailureResponse(InviteDialog dialog, int code, String reason, Message msg), called Wwhen a 400-699 failure response is received for a Re-INVITE transaction;
- onDlgReInviteTimeout(InviteDialog dialog), called when a Re-INVITE transaction expires;
- onDlgAck(InviteDialog dialog, String body, Message msg), called when an incoming ACK is received for an INVITE transaction;
- onDlgCall(InviteDialog dialog), called when the INVITE handshake is successful terminated;
- onDlgCancel(InviteDialog dialog, Message msg), called when an incoming CANCEL is received for an INVITE transaction;
- onDlgBye(InviteDialog dialog, Message msg), called when an incoming BYE is received;
- onDlgByeResponse(InviteDialog dialog, Message msg), called when a final response is received for a Bye request;
- onDlgClose(InviteDialog dialog), called when the dialog is finally closed;

Finally, CallListener implements the following methods:
- onCallIncoming(Call call, NameAddress caller, String sdp, Message invite), called when arriving a new INVITE method (incoming call);
- public void onCallModifying(Call call, String sdp, Message invite), called when arriving a new Re-INVITE method (re-inviting/call modify);
- onCallRinging(Call call, Message resp), called when arriving a 180 Ringing;
- onCallAccepted(Call call, String sdp, Message resp), called when arriving a 2xx (call accepted);
- onCallRefused(Call call, String reason, Message resp), called when arriving a 4xx (call failure);
- onCallRedirection(Call call, String reason, Vector contact_list, Message resp), called when arriving a 3xx (call redirection);
- onCallConfirmed(Call call, String sdp, Message ack), called when arriving an ACK method (call confirmed);
- onCallTimeout(Call call), called when the invite expires;
- onCallReInviteAccepted(Call call, String sdp, Message resp), called when arriving a 2xx (re-invite/modify accepted);
- onCallReInviteRefused(Call call, String reason, Message resp), called when arriving a 4xx (re-invite/modify failure);
- onCallReInviteTimeout(Call call), called when a re-invite expires;
- onCallCanceling(Call call, Message cancel), function called when arriving a CANCEL method (cancel request);
- onCallClosing(Call call, Message bye), called when arriving a BYE method (close request);
- onCallClosed(Call call, Message resp), called when arriving a 200 OK after a BYE request (call closed);

Methods *call()*, *listen()*, *accept()* can be used to initiate a outgoing or incoming calls, while method *handgup()* is used to teardown the calls.

Please refer to the javadoc documentation for all API details.

9. Messages, Headers, and other stuffs

[TO DO]

…

10. SipStack configuration

Although objects of types SipProvider, Transactions, UAs, and Servers can be configured through their specific constructors and ‘set’ methods, they can be also easily configured by appropriate configuration files that are passed to their constructors (actually a new Configure(file_name) object is used as constructor’s parameter). Through a configuration file you can also set all inner stack parameters such as timeouts, old RFC compliant behaviors, and log-handling settings.
For this purpose you can use several separated configuration files (one for each object/layer) or you can put your preferences all in one file.
Hereafter you can find the description of all configuration parameters, listed into different sections of one configuration file (it is up to you to split it into multiple files or not).
Note that there is a class named **SipStack** that includes all static stack attributes, and that is configured only one time for all SIP applications that start from the same program (i.e. instantiated by the same main class). For this reason be sure that the desired parameter are placed in the first configuration file you passes to you implementation.
On the other hand, all other non-static parameters can be loaded individually for each object instantiation (one or mode SipProviders, one or more Servers, etc.).

Here is the complete set of configuration parameters:

10.1. MjSip-1.4.1 Configuration

```
# MjSip parameters are organized into 5 sections:
#  o Section 1: SipStack base configuration
#  o Section 2: Logs
#  o Section 3: SipProvider configuration
#  o Section 4: UA/3PCC configuration
#  o Section 5: Server configuration
#


# ________________________________________________________________
#
# Section 1: SipStack base configuration
# ________________________________________________________________
#
# Normally, you do not have to change the base configuration,
# and you can go directly to Section 2.
# SIP and transport layer configurations are handled in Section 3.
#

# Default SIP port
# Default value: default_port=5060
#default_port=5060

# Default supported transport protocols
# Default value: default_transport_protocols=udp,tcp
#default_transport_protocols=udp

# Default max number of contemporary open transport connections
# Default value: default_nmax_connections=32
#default_nmax_connections=0

# Whether adding 'rport' parameter by default.
# Default value: use_rport=yes
#use_rport=no

# Default max-forwards value (RFC3261 recommends value 70)
# Default value: max_forwards=70
#max_forwards=10

# Starting retransmission timeout (milliseconds); called T1 in RFC2361; they suggest T1=500ms
# Default value: retransmission_timeout=500
#retransmission_timeout=2000

# Maximum retransmission timeout (milliseconds); called T2 in RFC2361; they suggest T2=4sec
# Default value: max_retransmission_timeout=4000
#max_retransmission_timeout=4000
```

```
# Transaction timeout (milliseconds); RFC2361 suggests 64*T1=32000ms
# Default value: transaction_timeout=32000
#transaction_timeout=10000

# Clearing timeout (milliseconds); T4 in RFC2361; they suggest T4=5sec
# Default value: clearing_timeout=5000
#clearing_timeout=5000

# Whether 1xx responses create an "early dialog" for methods that create dialog
# Default value: early_dialog=no
#early_dialog=yes

# Default 'expires' value in seconds. RFC2361 gives as default value expires=3600
# Default value: expires=3600
#expires=1800

# UA info included in request messages (in the User-Agent header field)
# Use 'NO-UA-INFO' string or let it blank if the User-Agent header filed must be added
# Default: ua_info=<the mjsip release>
# ua_info=NO-UA-INFO

# Server info included in request messages (in the Server header field)
# Use 'NO-SERVER-INFO' string or let it blank if the Server header filed must be added
# Default: server_info=<the mjsip release>
# server_info=NO-SERVER-INFO


# _________________________________________________________________
#
# Section 2: Logs
# _________________________________________________________________
#
# Change these parameters in order to customize how log-files are handled.
# By default log files are placed into the ./log folder, they are not rotated,
# and the maximum size is 2M.
#

# Log level. Only logs with a level less or equal to this are written
# Default value: debug_level=3
#debug_level=0

# Path for the log folder where log files are written
# By default, it is used the "./log" folder
# Use ".", to store logs in the root folder
# Default value: log_path=./log
#log_path= .

#The size limit of the log file [kB]
# Default value: max_logsize=2048
#max_logsize=4096

# The number of rotations of log files. Use '0' for NO rotation, '1' for rotating a single file
# Default value: log_rotations=0
#log_rotations=4

# The rotation period in MONTHs, DAYs, HOURs, or MINUTEs
# example: "log_rotation_time=3 MONTHS", that is equivalent to "log_rotations=90 DAYS"
# Default value: log_rotation_time=2 MONTHS
#log_rotation_time=7 DAYS


# _________________________________________________________________
#
# Section 3: SipProvider configuration
# _________________________________________________________________
#
# Change these parameters in order to customize the SIP transport layer.
# Usually you have to deal with some of these configuration parameters.
#

# Via address/name
# Use 'AUTO-CONFIGURATION' for auto detection, or let it undefined
# Default value: host_addr=AUTO-CONFIGURATION
#host_addr=192.168.0.33

# Local SIP port
# Default value: host_port=5060
#host_port=5060
```

```
# Network interface (IP address) used by SIP
# Use 'all-interfaces' for binding SIP to all interfaces (or let it undefined)
# Default value: host_ifaddr=ALL-INTERFACES
#host_ifaddr=192.168.0.33

# List of enabled transport protocols (the first protocol is used as default)
# Default value: transport_protocols=udp
#transport_protocols=udp,tcp

# Max number of contemporary open transport connections
# Default value: nmax_connections=32
#nmax_connections=0

# Outbound proxy
# Use 'NO-OUTBOUND' for not using an outbound proxy (or let it undefined)
# Default value: outbound_addr=NO-OUTBOUND
#outbound_addr=proxy.wonderland.net

# Port number of the outbound proxy
# Default value: outbound_port=5060
#outbound_port=5060


# _________________________________________________________________
#
# Section 4: UA/3PCC configuration
# _________________________________________________________________
#
# Change these parameters in order to customize the UA profile.
# You need to edit this section only if you are using a MjSip UA or
# you are managing 3PCC services.
#

# User's URL (From URL)
# If not defined (default), it equals the contact_url
#from_url=sip:alice@wonderland.net

# Contact URL
# If not defined (default), it is formed by sip:local_user@host_address:host_port
#contact_url=sip:alice@192.168.0.55:5070

# Local user name (used to build the contact url if not explitely defined)
# Default value: local_user=user
#contact_user=alice

# Path for the 'ua.jar' lib, used to retrive various UA media (gif, wav, etc.)
# By default, it is used the "lib/ua.jar" folder
#ua_jar=./ua.jar

# Path for the 'contacts.lst' file where save and load the list of VisualUA contacts
# By default, it is used the "contacts.lst" folder
#contacts_file=config/contacts.lst

# Whether using JMF for audio/video streaming
# Default value: use_jmf=no
#use_jmf=yes

# Whether using RAT (Robust Audio Tool) as audio sender/receiver
# Default value: use_rat=no
#use_rat=yes

# RAT command-line executable
# Default value: bin_rat=rat
#bin_rat=c:\programmi\mbone\rat

# Whether using VIC (Video Conferencing Tool) as video sender/receiver
# Default value: use_vic=no
#use_vic=yes

# VIC command-line executable
# Default value: bin_vic=vic
#bin_vic=c:\programmi\mbone\rat



# _________________________________________________________________
#
```

```
# Section 5: Server configuration
# ________________________________________________________________
#
# Change these parameters in order to customize the Server behaviour.
# You need to edit this section only if you are using a MjSip Server.
#

# The domain names that the server administers
# It lists the domain names for which the Location Service wmaintains user bindings.
# Use 'auto-configuration' for auto domanin name configuration (default).
domain_names=wonderland.net biloxi.example.com

# Whether consider any port as valid local local domain port
# (regardless which sip port is used).
# Default value: domain_port_any=no
#domain_port_any=yes

# The LocationService DB name
# Default value: location_db=users.db
#location_db=config/users.db

# Whether LocationService DB has to be cleaned at startup
# Default value: location_db_clean=no
#location_db_clean=yes

# Whether the Server should act as Registrar (i.e. respond to REGISTER requests).
# Default value: is_registrar=yes
#is_registrar=no

# Whether the Registrar can register new users (i.e. REGISTER requests from unregistered users).
# Default value: register_new_users=yes
#register_new_users=no

# Whether the server should stay in the signaling path (uses Record-Route/Route))
# Default value: on_route=no
#on_route=yes

# Whether refer to the RFC3261 Loose Route (or RFC2543 Strict Route) rule
# Default value: loose_route=yes
#loose_route=no
```

11. Examples

[TO DO]

…

12. References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, “ SIP: Session Initiation Protocol”, IETF RFC 3261, June 2002

[2] IETF SIP WG, http://www.ietf.org/html.charters/sip-charter.html

[3] SIP Page at Columbia University, http://www.cs.columbia.edu/sip

[4] M. Handley, H. Schulzrinne, J. Rosenberg, “SIP: Session Initiation Protocol”, IETF RFC 2543, May 1999, Obsolated by RFC 3261

[5] M. Handley, V. Jacobson, “SDP: Session Description Protocol”, IETF RFC 2327, April 1998

[6] J. Rosenberg, H. Schulzrinne, “An Offer/Answer Model with the Session Description Protocol (SDP)”, IETF RFC 3264, June 2002