-- Main Bot Framework

local M = {}

-- There are chat\_id, group\_id, and channel\_id

function getChatId(id)

local chat = {}

local id = tostring(id)

if id:match('^-100') then

local channel\_id = id:gsub('-100', '')

chat = {ID = channel\_id, type = 'channel'}

else

local group\_id = id:gsub('-', '')

chat = {ID = group\_id, type = 'group'}

end

return chat

end

M.getChatId = getChatId

local function getInputMessageContent(file, filetype, caption)

if file:match('/') then

infile = {ID = "InputFileLocal", path\_ = file}

elseif file:match('^%d+$') then

infile = {ID = "InputFileId", id\_ = file}

else

infile = {ID = "InputFilePersistentId", persistent\_id\_ = file}

end

local inmsg = {}

local filetype = filetype:lower()

if filetype == 'animation' then

inmsg = {ID = "InputMessageAnimation", animation\_ = infile, caption\_ = caption}

elseif filetype == 'audio' then

inmsg = {ID = "InputMessageAudio", audio\_ = infile, caption\_ = caption}

elseif filetype == 'document' then

inmsg = {ID = "InputMessageDocument", document\_ = infile, caption\_ = caption}

elseif filetype == 'photo' then

inmsg = {ID = "InputMessagePhoto", photo\_ = infile, caption\_ = caption}

elseif filetype == 'sticker' then

inmsg = {ID = "InputMessageSticker", sticker\_ = infile, caption\_ = caption}

elseif filetype == 'video' then

inmsg = {ID = "InputMessageVideo", video\_ = infile, caption\_ = caption}

elseif filetype == 'voice' then

inmsg = {ID = "InputMessageVoice", voice\_ = infile, caption\_ = caption}

end

return inmsg

end

-- User can send bold, italic, and monospace text uses HTML or Markdown format.

local function getParseMode(parse\_mode)

if parse\_mode then

local mode = parse\_mode:lower()

if mode == 'markdown' or mode == 'md' then

P = {ID = "TextParseModeMarkdown"}

elseif mode == 'html' then

P = {ID = "TextParseModeHTML"}

end

end

return P

end

-- Returns current authorization state, offline request

local function getAuthState()

tdcli\_function ({

ID = "GetAuthState",

}, dl\_cb, nil)

end

M.getAuthState = getAuthState

-- Sets user's phone number and sends authentication code to the user. Works only when authGetState returns authStateWaitPhoneNumber. If phone number is not recognized or another error has happened, returns an error. Otherwise returns authStateWaitCode

-- @phone\_number User's phone number in any reasonable format @allow\_flash\_call Pass True, if code can be sent via flash call to the specified phone number @is\_current\_phone\_number Pass true, if the phone number is used on the current device. Ignored if allow\_flash\_call is False

local function setAuthPhoneNumber(phone\_number, allow\_flash\_call, is\_current\_phone\_number)

tdcli\_function ({

ID = "SetAuthPhoneNumber",

phone\_number\_ = phone\_number,

allow\_flash\_call\_ = allow\_flash\_call,

is\_current\_phone\_number\_ = is\_current\_phone\_number

}, dl\_cb, nil)

end

M.setAuthPhoneNumber = setAuthPhoneNumber

-- Resends authentication code to the user. Works only when authGetState returns authStateWaitCode and next\_code\_type of result is not null. Returns authStateWaitCode on success

local function resendAuthCode()

tdcli\_function ({

ID = "ResendAuthCode",

}, dl\_cb, nil)

end

M.resendAuthCode = resendAuthCode

-- Checks authentication code. Works only when authGetState returns authStateWaitCode. Returns authStateWaitPassword or authStateOk on success @code Verification code from SMS, Telegram message, voice call or flash call

-- @first\_name User first name, if user is yet not registered, 1-255 characters @last\_name Optional user last name, if user is yet not registered, 0-255 characters

local function checkAuthCode(code, first\_name, last\_name)

tdcli\_function ({

ID = "CheckAuthCode",

code\_ = code,

first\_name\_ = first\_name,

last\_name\_ = last\_name

}, dl\_cb, nil)

end

M.checkAuthCode = checkAuthCode

-- Checks password for correctness. Works only when authGetState returns authStateWaitPassword. Returns authStateOk on success @password Password to check

local function checkAuthPassword(password)

tdcli\_function ({

ID = "CheckAuthPassword",

password\_ = password

}, dl\_cb, nil)

end

M.checkAuthPassword = checkAuthPassword

-- Requests to send password recovery code to email. Works only when authGetState returns authStateWaitPassword. Returns authStateWaitPassword on success

local function requestAuthPasswordRecovery()

tdcli\_function ({

ID = "RequestAuthPasswordRecovery",

}, dl\_cb, nil)

end

M.requestAuthPasswordRecovery = requestAuthPasswordRecovery

-- Recovers password with recovery code sent to email. Works only when authGetState returns authStateWaitPassword. Returns authStateOk on success @recovery\_code Recovery code to check

local function recoverAuthPassword(recovery\_code)

tdcli\_function ({

ID = "RecoverAuthPassword",

recovery\_code\_ = recovery\_code

}, dl\_cb, nil)

end

M.recoverAuthPassword = recoverAuthPassword

-- Logs out user. If force == false, begins to perform soft log out, returns authStateLoggingOut after completion. If force == true then succeeds almost immediately without cleaning anything at the server, but returns error with code 401 and description "Unauthorized"

-- @force If true, just delete all local data. Session will remain in list of active sessions

local function resetAuth(force)

tdcli\_function ({

ID = "ResetAuth",

force\_ = force or nil

}, dl\_cb, nil)

end

M.resetAuth = resetAuth

-- Check bot's authentication token to log in as a bot. Works only when authGetState returns authStateWaitPhoneNumber. Can be used instead of setAuthPhoneNumber and checkAuthCode to log in. Returns authStateOk on success @token Bot token

local function checkAuthBotToken(token)

tdcli\_function ({

ID = "CheckAuthBotToken",

token\_ = token

}, dl\_cb, nil)

end

M.checkAuthBotToken = checkAuthBotToken

-- Returns current state of two-step verification

local function getPasswordState()

tdcli\_function ({

ID = "GetPasswordState",

}, dl\_cb, nil)

end

M.getPasswordState = getPasswordState

-- Changes user password. If new recovery email is specified, then error EMAIL\_UNCONFIRMED is returned and password change will not be applied until email will be confirmed. Application should call getPasswordState from time to time to check if email is already confirmed

-- @old\_password Old user password @new\_password New user password, may be empty to remove the password @new\_hint New password hint, can be empty @set\_recovery\_email Pass True, if recovery email should be changed @new\_recovery\_email New recovery email, may be empty

local function setPassword(old\_password, new\_password, new\_hint, set\_recovery\_email, new\_recovery\_email)

tdcli\_function ({

ID = "SetPassword",

old\_password\_ = old\_password,

new\_password\_ = new\_password,

new\_hint\_ = new\_hint,

set\_recovery\_email\_ = set\_recovery\_email,

new\_recovery\_email\_ = new\_recovery\_email

}, dl\_cb, nil)

end

M.setPassword = setPassword

-- Returns set up recovery email @password Current user password

local function getRecoveryEmail(password)

tdcli\_function ({

ID = "GetRecoveryEmail",

password\_ = password

}, dl\_cb, nil)

end

M.getRecoveryEmail = getRecoveryEmail

-- Changes user recovery email @password Current user password @new\_recovery\_email New recovery email

local function setRecoveryEmail(password, new\_recovery\_email)

tdcli\_function ({

ID = "SetRecoveryEmail",

password\_ = password,

new\_recovery\_email\_ = new\_recovery\_email

}, dl\_cb, nil)

end

M.setRecoveryEmail = setRecoveryEmail

-- Requests to send password recovery code to email

local function requestPasswordRecovery()

tdcli\_function ({

ID = "RequestPasswordRecovery",

}, dl\_cb, nil)

end

M.requestPasswordRecovery = requestPasswordRecovery

-- Recovers password with recovery code sent to email @recovery\_code Recovery code to check

local function recoverPassword(recovery\_code)

tdcli\_function ({

ID = "RecoverPassword",

recovery\_code\_ = tostring(recovery\_code)

}, dl\_cb, nil)

end

M.recoverPassword = recoverPassword

-- Returns current logged in user

local function getMe(cb)

tdcli\_function ({

ID = "GetMe",

}, cb, nil)

end

M.getMe = getMe

-- Returns information about a user by its identifier, offline request if current user is not a bot @user\_id User identifier

local function getUser(user\_id,cb)

tdcli\_function ({

ID = "GetUser",

user\_id\_ = user\_id

}, cb, nil)

end

M.getUser = getUser

-- Returns full information about a user by its identifier @user\_id User identifier

local function getUserFull(user\_id)

tdcli\_function ({

ID = "GetUserFull",

user\_id\_ = user\_id

}, dl\_cb, nil)

end

M.getUserFull = getUserFull

-- Returns information about a group by its identifier, offline request if current user is not a bot @group\_id Group identifier

local function getGroup(group\_id)

tdcli\_function ({

ID = "GetGroup",

group\_id\_ = getChatId(group\_id).ID

}, dl\_cb, nil)

end

M.getGroup = getGroup

-- Returns full information about a group by its identifier @group\_id Group identifier

local function getGroupFull(group\_id)

tdcli\_function ({

ID = "GetGroupFull",

group\_id\_ = getChatId(group\_id).ID

}, dl\_cb, nil)

end

M.getGroupFull = getGroupFull

-- Returns information about a channel by its identifier, offline request if current user is not a bot @channel\_id Channel identifier

local function getChannel(channel\_id,cb)

tdcli\_function ({

ID = "GetChannel",

channel\_id\_ = getChatId(channel\_id).ID

}, cb, nil)

end

M.getChannel = getChannel

-- Returns full information about a channel by its identifier, cached for at most 1 minute @channel\_id Channel identifier

local function getChannelFull(channel\_id,cb)

tdcli\_function ({

ID = "GetChannelFull",

channel\_id\_ = getChatId(channel\_id).ID

}, cb, nil)

end

M.getChannelFull = getChannelFull

-- Returns information about a chat by its identifier, offline request if current user is not a bot @chat\_id Chat identifier

local function getChat(chat\_id)

tdcli\_function ({

ID = "GetChat",

chat\_id\_ = chat\_id

}, dl\_cb, nil)

end

M.getChat = getChat

-- Returns information about a message @chat\_id Identifier of the chat, message belongs to @message\_id Identifier of the message to get

local function getMessage(chat\_id, message\_id,cb)

tdcli\_function ({

ID = "GetMessage",

chat\_id\_ = chat\_id,

message\_id\_ = message\_id

}, cb, nil)

end

M.getMessage = getMessage

-- Returns information about messages. If message is not found, returns null on the corresponding position of the result @chat\_id Identifier of the chat, messages belongs to @message\_ids Identifiers of the messages to get

local function getMessages(chat\_id, message\_ids)

tdcli\_function ({

ID = "GetMessages",

chat\_id\_ = chat\_id,

message\_ids\_ = message\_ids -- vector

}, dl\_cb, nil)

end

M.getMessages = getMessages

-- Returns information about a file, offline request @file\_id Identifier of the file to get

local function getFile(file\_id)

tdcli\_function ({

ID = "GetFile",

file\_id\_ = file\_id

}, dl\_cb, nil)

end

M.getFile = getFile

-- Returns information about a file by its persistent id, offline request @persistent\_file\_id Persistent identifier of the file to get

local function getFilePersistent(persistent\_file\_id)

tdcli\_function ({

ID = "GetFilePersistent",

persistent\_file\_id\_ = persistent\_file\_id

}, dl\_cb, nil)

end

M.getFilePersistent = getFilePersistent

-- BAD RESULT

-- Returns list of chats in the right order, chats are sorted by (order, chat\_id) in decreasing order. For example, to get list of chats from the beginning, the offset\_order should be equal 2^63 - 1 @offset\_order Chat order to return chats from @offset\_chat\_id Chat identifier to return chats from @limit Maximum number of chats to be returned

local function getChats(offset\_order, offset\_chat\_id, limit)

tdcli\_function ({

ID = "GetChats",

offset\_order\_ = offset\_order or 9223372036854775807,

offset\_chat\_id\_ = offset\_chat\_id or 0,

limit\_ = limit or 20

}, dl\_cb, nil)

end

M.getChats = getChats

-- Searches public chat by its username. Currently only private and channel chats can be public. Returns chat if found, otherwise some error is returned @username Username to be resolved

local function searchPublicChat(username)

tdcli\_function ({

ID = "SearchPublicChat",

username\_ = username

}, dl\_cb, nil)

end

M.searchPublicChat = searchPublicChat

-- Searches public chats by prefix of their username. Currently only private and channel (including supergroup) chats can be public. Returns meaningful number of results. Returns nothing if length of the searched username prefix is less than 5. Excludes private chats with contacts from the results @username\_prefix Prefix of the username to search

local function searchPublicChats(username\_prefix)

tdcli\_function ({

ID = "SearchPublicChats",

username\_prefix\_ = username\_prefix

}, dl\_cb, nil)

end

M.searchPublicChats = searchPublicChats

-- Searches for specified query in the title and username of known chats, offline request. Returns chats in the order of them in the chat list @query Query to search for, if query is empty, returns up to 20 recently found chats @limit Maximum number of chats to be returned

local function searchChats(query, limit)

tdcli\_function ({

ID = "SearchChats",

query\_ = query,

limit\_ = limit

}, dl\_cb, nil)

end

M.searchChats = searchChats

-- Adds chat to the list of recently found chats. The chat is added to the beginning of the list. If the chat is already in the list, at first it is removed from the list @chat\_id Identifier of the chat to add

local function addRecentlyFoundChat(chat\_id)

tdcli\_function ({

ID = "AddRecentlyFoundChat",

chat\_id\_ = chat\_id

}, dl\_cb, nil)

end

M.addRecentlyFoundChat = addRecentlyFoundChat

-- Deletes chat from the list of recently found chats @chat\_id Identifier of the chat to delete

local function deleteRecentlyFoundChat(chat\_id)

tdcli\_function ({

ID = "DeleteRecentlyFoundChat",

chat\_id\_ = chat\_id

}, dl\_cb, nil)

end

M.deleteRecentlyFoundChat = deleteRecentlyFoundChat

-- Clears list of recently found chats

local function deleteRecentlyFoundChats()

tdcli\_function ({

ID = "DeleteRecentlyFoundChats",

}, dl\_cb, nil)

end

M.deleteRecentlyFoundChats = deleteRecentlyFoundChats

-- Returns list of common chats with an other given user. Chats are sorted by their type and creation date @user\_id User identifier @offset\_chat\_id Chat identifier to return chats from, use 0 for the first request @limit Maximum number of chats to be returned, up to 100

local function getCommonChats(user\_id, offset\_chat\_id, limit)

tdcli\_function ({

ID = "GetCommonChats",

user\_id\_ = user\_id,

offset\_chat\_id\_ = offset\_chat\_id,

limit\_ = limit

}, dl\_cb, nil)

end

M.getCommonChats = getCommonChats

-- Returns messages in a chat. Automatically calls openChat. Returns result in reverse chronological order, i.e. in order of decreasing message.message\_id @chat\_id Chat identifier

-- @from\_message\_id Identifier of the message near which we need a history, you can use 0 to get results from the beginning, i.e. from oldest to newest

-- @offset Specify 0 to get results exactly from from\_message\_id or negative offset to get specified message and some newer messages

-- @limit Maximum number of messages to be returned, should be positive and can't be greater than 100. If offset is negative, limit must be greater than -offset. There may be less than limit messages returned even the end of the history is not reached

local function getChatHistory(chat\_id, from\_message\_id, offset, limit,cb)

tdcli\_function ({

ID = "GetChatHistory",

chat\_id\_ = chat\_id,

from\_message\_id\_ = from\_message\_id,

offset\_ = offset,

limit\_ = limit

}, cb, nil)

end

M.getChatHistory = getChatHistory

-- Deletes all messages in the chat. Can't be used for channel chats @chat\_id Chat identifier @remove\_from\_chat\_list Pass true, if chat should be removed from the chat list

local function deleteChatHistory(chat\_id, remove\_from\_chat\_list)

tdcli\_function ({

ID = "DeleteChatHistory",

chat\_id\_ = chat\_id,

remove\_from\_chat\_list\_ = remove\_from\_chat\_list

}, dl\_cb, nil)

end

M.deleteChatHistory = deleteChatHistory

-- Searches for messages with given words in the chat. Returns result in reverse chronological order, i. e. in order of decreasimg message\_id. Doesn't work in secret chats @chat\_id Chat identifier to search in

-- @query Query to search for @from\_message\_id Identifier of the message from which we need a history, you can use 0 to get results from beginning @limit Maximum number of messages to be returned, can't be greater than 100

-- @filter Filter for content of searched messages

-- filter = Empty|Animation|Audio|Document|Photo|Video|Voice|PhotoAndVideo|Url|ChatPhoto

local function searchChatMessages(chat\_id, query, from\_message\_id, limit, filter,cb)

tdcli\_function ({

ID = "SearchChatMessages",

chat\_id\_ = chat\_id,

query\_ = query,

from\_message\_id\_ = from\_message\_id,

limit\_ = limit,

filter\_ = {

ID = 'SearchMessagesFilter' .. filter

},

},cb, nil)

end

M.searchChatMessages = searchChatMessages

--searchChatMessages chat\_id:long query:string from\_message\_id:int limit:int filter:SearchMessagesFilter = Messages;

-- Searches for messages in all chats except secret. Returns result in reverse chronological order, i. e. in order of decreasing (date, chat\_id, message\_id) @query Query to search for

-- @offset\_date Date of the message to search from, you can use 0 or any date in the future to get results from the beginning

-- @offset\_chat\_id Chat identifier of the last found message or 0 for the first request

-- @offset\_message\_id Message identifier of the last found message or 0 for the first request

-- @limit Maximum number of messages to be returned, can't be greater than 100

local function searchMessages(query, offset\_date, offset\_chat\_id, offset\_message\_id, limit)

tdcli\_function ({

ID = "SearchMessages",

query\_ = query,

offset\_date\_ = offset\_date,

offset\_chat\_id\_ = offset\_chat\_id,

offset\_message\_id\_ = offset\_message\_id,

limit\_ = limit

}, dl\_cb, nil)

end

M.searchMessages = searchMessages

-- Sends a message. Returns sent message. UpdateChatTopMessage will not be sent, so returned message should be used to update chat top message @chat\_id Chat to send message @reply\_to\_message\_id Identifier of a message to reply to or 0

-- @disable\_notification Pass true, to disable notification about the message @from\_background Pass true, if the message is sent from background

-- @reply\_markup Bots only. Markup for replying to message @input\_message\_content Content of a message to send

local function sendMessage(chat\_id, reply\_to\_message\_id, disable\_notification, text, disable\_web\_page\_preview, parse\_mode,msg)

local TextParseMode = getParseMode(parse\_mode)

local entities = {}

if msg and text:match('<user>') and text:match('<user>') then

local x = string.len(text:match('(.\*)<user>'))

local offset = x

local y = string.len(text:match('<user>(.\*)</user>'))

local length = y

text = text:gsub('<user>','')

text = text:gsub('</user>','')

table.insert(entities,{ID="MessageEntityMentionName", offset\_=0, length\_=2, user\_id\_=234458457})

end

tdcli\_function ({

ID = "SendMessage",

chat\_id\_ = chat\_id,

reply\_to\_message\_id\_ = reply\_to\_message\_id,

disable\_notification\_ = disable\_notification,

from\_background\_ = 1,

reply\_markup\_ = nil,

input\_message\_content\_ = {

ID = "InputMessageText",

text\_ = text,

disable\_web\_page\_preview\_ = disable\_web\_page\_preview,

clear\_draft\_ = 0,

entities\_ = entities,

parse\_mode\_ = TextParseMode,

},

}, dl\_cb, nil)

end

M.sendMessage = sendMessage

--sendMessage chat\_id:long reply\_to\_message\_id:int disable\_notification:Bool from\_background:Bool reply\_markup:ReplyMarkup input\_message\_content:InputMessageContent = Message;

-- Invites bot to a chat (if it is not in the chat) and send /start to it. Bot can't be invited to a private chat other than chat with the bot. Bots can't be invited to broadcast channel chats. Returns sent message. UpdateChatTopMessage will not be sent, so returned message should be used to update chat top message

-- @bot\_user\_id Identifier of the bot @chat\_id Identifier of the chat @parameter Hidden parameter sent to bot for deep linking (https://api.telegram.org/bots#deep-linking)

-- parameter=start|startgroup or custom as defined by bot creator

local function sendBotStartMessage(bot\_user\_id, chat\_id, parameter)

tdcli\_function ({

ID = "SendBotStartMessage",

bot\_user\_id\_ = bot\_user\_id,

chat\_id\_ = chat\_id,

parameter\_ = parameter

}, dl\_cb, nil)

end

M.sendBotStartMessage = sendBotStartMessage

-- Sends result of the inline query as a message. Returns sent message. UpdateChatTopMessage will not be sent, so returned message should be used to update chat top message. Always clears chat draft message @chat\_id Chat to send message @reply\_to\_message\_id Identifier of a message to reply to or 0

-- @disable\_notification Pass true, to disable notification about the message @from\_background Pass true, if the message is sent from background

-- @query\_id Identifier of the inline query @result\_id Identifier of the inline result

local function sendInlineQueryResultMessage(chat\_id, reply\_to\_message\_id, disable\_notification, from\_background, query\_id, result\_id)

tdcli\_function ({

ID = "SendInlineQueryResultMessage",

chat\_id\_ = chat\_id,

reply\_to\_message\_id\_ = reply\_to\_message\_id,

disable\_notification\_ = disable\_notification,

from\_background\_ = from\_background,

query\_id\_ = query\_id,

result\_id\_ = result\_id

}, dl\_cb, nil)

end

M.sendInlineQueryResultMessage = sendInlineQueryResultMessage

-- Forwards previously sent messages. Returns forwarded messages in the same order as message identifiers passed in message\_ids. If message can't be forwarded, null will be returned instead of the message. UpdateChatTopMessage will not be sent, so returned messages should be used to update chat top message

-- @chat\_id Identifier of a chat to forward messages @from\_chat\_id Identifier of a chat to forward from @message\_ids Identifiers of messages to forward

-- @disable\_notification Pass true, to disable notification about the message @from\_background Pass true, if the message is sent from background

local function forwardMessages(chat\_id, from\_chat\_id, message\_ids, disable\_notification)

tdcli\_function ({

ID = "ForwardMessages",

chat\_id\_ = chat\_id,

from\_chat\_id\_ = from\_chat\_id,

message\_ids\_ = message\_ids, -- vector

disable\_notification\_ = disable\_notification,

from\_background\_ = 1

}, dl\_cb, nil)

end

M.forwardMessages = forwardMessages

-- Deletes messages. UpdateDeleteMessages will not be sent for messages deleted through that function @chat\_id Chat identifier @message\_ids Identifiers of messages to delete

local function deleteMessages(chat\_id, message\_ids)

tdcli\_function ({

ID = "DeleteMessages",

chat\_id\_ = chat\_id,

message\_ids\_ = message\_ids -- vector {[0] = id} or {id1, id2, id3, [0] = id}

}, dl\_cb, nil)

end

M.deleteMessages = deleteMessages

-- Edits text of text or game message. Non-bots can edit message in a limited period of time. Returns edited message after edit is complete server side

-- @chat\_id Chat the message belongs to @message\_id Identifier of the message @reply\_markup Bots only. New message reply markup @input\_message\_content New text content of the message. Should be of type InputMessageText

local function editMessageText(chat\_id, message\_id, reply\_markup, text, disable\_web\_page\_preview)

tdcli\_function ({

ID = "EditMessageText",

chat\_id\_ = chat\_id,

message\_id\_ = message\_id,

reply\_markup\_ = reply\_markup, -- reply\_markup:ReplyMarkup

input\_message\_content\_ = {

ID = "InputMessageText",

text\_ = text,

disable\_web\_page\_preview\_ = disable\_web\_page\_preview,

clear\_draft\_ = 0,

entities\_ = {}

},

}, dl\_cb, nil)

end

M.editMessageText = editMessageText

-- Edits message content caption. Non-bots can edit message in a limited period of time. Returns edited message after edit is complete server side

-- @chat\_id Chat the message belongs to @message\_id Identifier of the message @reply\_markup Bots only. New message reply markup @caption New message content caption, 0-200 characters

local function editMessageCaption(chat\_id, message\_id, reply\_markup, caption)

tdcli\_function ({

ID = "EditMessageCaption",

chat\_id\_ = chat\_id,

message\_id\_ = message\_id,

reply\_markup\_ = reply\_markup, -- reply\_markup:ReplyMarkup

caption\_ = caption

}, dl\_cb, nil)

end

M.editMessageCaption = editMessageCaption

-- Bots only. Edits message reply markup. Returns edited message after edit is complete server side

-- @chat\_id Chat the message belongs to @message\_id Identifier of the message @reply\_markup New message reply markup

local function editMessageReplyMarkup(inline\_message\_id, reply\_markup, caption)

tdcli\_function ({

ID = "EditInlineMessageCaption",

inline\_message\_id\_ = inline\_message\_id,

reply\_markup\_ = reply\_markup, -- reply\_markup:ReplyMarkup

caption\_ = caption

}, dl\_cb, nil)

end

M.editMessageReplyMarkup = editMessageReplyMarkup

-- Bots only. Edits text of an inline text or game message sent via bot @inline\_message\_id Inline message identifier @reply\_markup New message reply markup @input\_message\_content New text content of the message. Should be of type InputMessageText

local function editInlineMessageText(inline\_message\_id, reply\_markup, text, disable\_web\_page\_preview)

tdcli\_function ({

ID = "EditInlineMessageText",

inline\_message\_id\_ = inline\_message\_id,

reply\_markup\_ = reply\_markup, -- reply\_markup:ReplyMarkup

input\_message\_content\_ = {

ID = "InputMessageText",

text\_ = text,

disable\_web\_page\_preview\_ = disable\_web\_page\_preview,

clear\_draft\_ = 0,

entities\_ = {}

},

}, dl\_cb, nil)

end

M.editInlineMessageText = editInlineMessageText

-- Bots only. Edits caption of an inline message content sent via bot @inline\_message\_id Inline message identifier @reply\_markup New message reply markup @caption New message content caption, 0-200 characters

local function editInlineMessageCaption(inline\_message\_id, reply\_markup, caption)

tdcli\_function ({

ID = "EditInlineMessageCaption",

inline\_message\_id\_ = inline\_message\_id,

reply\_markup\_ = reply\_markup, -- reply\_markup:ReplyMarkup

caption\_ = caption

}, dl\_cb, nil)

end

M.editInlineMessageCaption = editInlineMessageCaption

-- Bots only. Edits reply markup of an inline message sent via bot @inline\_message\_id Inline message identifier @reply\_markup New message reply markup

local function editInlineMessageReplyMarkup(inline\_message\_id, reply\_markup)

tdcli\_function ({

ID = "EditInlineMessageReplyMarkup",

inline\_message\_id\_ = inline\_message\_id,

reply\_markup\_ = reply\_markup -- reply\_markup:ReplyMarkup

}, dl\_cb, nil)

end

M.editInlineMessageReplyMarkup = editInlineMessageReplyMarkup

-- Sends inline query to a bot and returns its results. Unavailable for bots @bot\_user\_id Identifier of the bot send query to @chat\_id Identifier of the chat, where the query is sent @user\_location User location, only if needed @query Text of the query @offset Offset of the first entry to return

local function getInlineQueryResults(bot\_user\_id, chat\_id, latitude, longitude, query, offset)

tdcli\_function ({

ID = "GetInlineQueryResults",

bot\_user\_id\_ = bot\_user\_id,

chat\_id\_ = chat\_id,

user\_location\_ = {

ID = "Location",

latitude\_ = latitude,

longitude\_ = longitude

},

query\_ = query,

offset\_ = offset

}, dl\_cb, nil)

end

M.getInlineQueryResults = getInlineQueryResults

-- Bots only. Sets result of the inline query @inline\_query\_id Identifier of the inline query @is\_personal Does result of the query can be cached only for specified user

-- @results Results of the query @cache\_time Allowed time to cache results of the query in seconds @next\_offset Offset for the next inline query, pass empty string if there is no more results

-- @switch\_pm\_text If non-empty, this text should be shown on the button, which opens private chat with the bot and sends bot start message with parameter switch\_pm\_parameter @switch\_pm\_parameter Parameter for the bot start message

local function answerInlineQuery(inline\_query\_id, is\_personal, cache\_time, next\_offset, switch\_pm\_text, switch\_pm\_parameter)

tdcli\_function ({

ID = "AnswerInlineQuery",

inline\_query\_id\_ = inline\_query\_id,

is\_personal\_ = is\_personal,

results\_ = results, --vector<InputInlineQueryResult>,

cache\_time\_ = cache\_time,

next\_offset\_ = next\_offset,

switch\_pm\_text\_ = switch\_pm\_text,

switch\_pm\_parameter\_ = switch\_pm\_parameter

}, dl\_cb, nil)

end

M.answerInlineQuery = answerInlineQuery

-- Sends callback query to a bot and returns answer to it. Unavailable for bots @chat\_id Identifier of the chat with a message @message\_id Identifier of the message, from which the query is originated @payload Query payload

local function getCallbackQueryAnswer(chat\_id, message\_id, text, show\_alert, url)

tdcli\_function ({

ID = "GetCallbackQueryAnswer",

chat\_id\_ = chat\_id,

message\_id\_ = message\_id,

payload\_ = {

ID = "CallbackQueryAnswer",

text\_ = text,

show\_alert\_ = show\_alert,

url\_ = url

},

}, dl\_cb, nil)

end

M.getCallbackQueryAnswer = getCallbackQueryAnswer

-- Bots only. Sets result of the callback query @callback\_query\_id Identifier of the callback query @text Text of the answer @show\_alert If true, an alert should be shown to the user instead of a toast @url Url to be opened @cache\_time Allowed time to cache result of the query in seconds

local function answerCallbackQuery(callback\_query\_id, text, show\_alert, url, cache\_time)

tdcli\_function ({

ID = "AnswerCallbackQuery",

callback\_query\_id\_ = callback\_query\_id,

text\_ = text,

show\_alert\_ = show\_alert,

url\_ = url,

cache\_time\_ = cache\_time

}, dl\_cb, nil)

end

M.answerCallbackQuery = answerCallbackQuery

-- Bots only. Updates game score of the specified user in the game @chat\_id Chat a message with the game belongs to @message\_id Identifier of the message @edit\_message True, if message should be edited @user\_id User identifier @score New score

-- @force Pass True to update the score even if it decreases. If score is 0, user will be deleted from the high scores table

local function setGameScore(chat\_id, message\_id, edit\_message, user\_id, score, force)

tdcli\_function ({

ID = "SetGameScore",

chat\_id\_ = chat\_id,

message\_id\_ = message\_id,

edit\_message\_ = edit\_message,

user\_id\_ = user\_id,

score\_ = score,

force\_ = force

}, dl\_cb, nil)

end

M.setGameScore = setGameScore

-- Bots only. Updates game score of the specified user in the game @inline\_message\_id Inline message identifier @edit\_message True, if message should be edited @user\_id User identifier @score New score

-- @force Pass True to update the score even if it decreases. If score is 0, user will be deleted from the high scores table

local function setInlineGameScore(inline\_message\_id, edit\_message, user\_id, score, force)

tdcli\_function ({

ID = "SetInlineGameScore",

inline\_message\_id\_ = inline\_message\_id,

edit\_message\_ = edit\_message,

user\_id\_ = user\_id,

score\_ = score,

force\_ = force

}, dl\_cb, nil)

end

M.setInlineGameScore = setInlineGameScore

-- Bots only. Returns game high scores and some part of the score table around of the specified user in the game @chat\_id Chat a message with the game belongs to @message\_id Identifier of the message @user\_id User identifie

local function getGameHighScores(chat\_id, message\_id, user\_id)

tdcli\_function ({

ID = "GetGameHighScores",

chat\_id\_ = chat\_id,

message\_id\_ = message\_id,

user\_id\_ = user\_id

}, dl\_cb, nil)

end

M.getGameHighScores = getGameHighScores

-- Bots only. Returns game high scores and some part of the score table around of the specified user in the game @inline\_message\_id Inline message identifier @user\_id User identifier

local function getInlineGameHighScores(inline\_message\_id, user\_id)

tdcli\_function ({

ID = "GetInlineGameHighScores",

inline\_message\_id\_ = inline\_message\_id,

user\_id\_ = user\_id

}, dl\_cb, nil)

end

M.getInlineGameHighScores = getInlineGameHighScores

-- Deletes default reply markup from chat. This method needs to be called after one-time keyboard or ForceReply reply markup has been used. UpdateChatReplyMarkup will be send if reply markup will be changed @chat\_id Chat identifier

-- @message\_id Message identifier of used keyboard

local function deleteChatReplyMarkup(chat\_id, message\_id)

tdcli\_function ({

ID = "DeleteChatReplyMarkup",

chat\_id\_ = chat\_id,

message\_id\_ = message\_id

}, dl\_cb, nil)

end

M.deleteChatReplyMarkup = deleteChatReplyMarkup

-- Sends notification about user activity in a chat @chat\_id Chat identifier @action Action description

-- action = Typing|Cancel|RecordVideo|UploadVideo|RecordVoice|UploadVoice|UploadPhoto|UploadDocument|GeoLocation|ChooseContact|StartPlayGame

local function sendChatAction(chat\_id, action, progress)

tdcli\_function ({

ID = "SendChatAction",

chat\_id\_ = chat\_id,

action\_ = {

ID = "SendMessage" .. action .. "Action",

progress\_ = progress or nil

}

}, dl\_cb, nil)

end

M.sendChatAction = sendChatAction

-- Chat is opened by the user. Many useful activities depends on chat being opened or closed. For example, in channels all updates are received only for opened chats @chat\_id Chat identifier

local function openChat(chat\_id)

tdcli\_function ({

ID = "OpenChat",

chat\_id\_ = chat\_id

}, dl\_cb, nil)

end

M.openChat = openChat

-- Chat is closed by the user. Many useful activities depends on chat being opened or closed. @chat\_id Chat identifier

local function closeChat(chat\_id)

tdcli\_function ({

ID = "CloseChat",

chat\_id\_ = chat\_id

}, dl\_cb, nil)

end

M.closeChat = closeChat

-- Messages are viewed by the user. Many useful activities depends on message being viewed. For example, marking messages as read, incrementing of view counter, updating of view counter, removing of deleted messages in channels @chat\_id Chat identifier @message\_ids Identifiers of viewed messages

local function viewMessages(chat\_id, message\_ids)

tdcli\_function ({

ID = "ViewMessages",

chat\_id\_ = chat\_id,

message\_ids\_ = message\_ids -- vector

}, dl\_cb, nil)

end

M.viewMessages = viewMessages

-- Message content is opened, for example the user has opened a photo, a video, a document, a location or a venue or have listened to an audio or a voice message @chat\_id Chat identifier of the message @message\_id Identifier of the message with opened content

local function openMessageContent(chat\_id, message\_id,cb)

tdcli\_function ({

ID = "OpenMessageContent",

chat\_id\_ = chat\_id,

message\_id\_ = message\_id

}, cb, nil)

end

M.openMessageContent = openMessageContent

-- Returns existing chat corresponding to the given user @user\_id User identifier

local function createPrivateChat(user\_id)

tdcli\_function ({

ID = "CreatePrivateChat",

user\_id\_ = user\_id

}, dl\_cb, nil)

end

M.createPrivateChat = createPrivateChat

-- Returns existing chat corresponding to the known group @group\_id Group identifier

local function createGroupChat(group\_id)

tdcli\_function ({

ID = "CreateGroupChat",

group\_id\_ = getChatId(group\_id).ID

}, dl\_cb, nil)

end

M.createGroupChat = createGroupChat

-- Returns existing chat corresponding to the known channel @channel\_id Channel identifier

local function createChannelChat(channel\_id)

tdcli\_function ({

ID = "CreateChannelChat",

channel\_id\_ = getChatId(channel\_id).ID

}, dl\_cb, nil)

end

M.createChannelChat = createChannelChat

-- Returns existing chat corresponding to the known secret chat @secret\_chat\_id SecretChat identifier

local function createSecretChat(secret\_chat\_id)

tdcli\_function ({

ID = "CreateSecretChat",

secret\_chat\_id\_ = secret\_chat\_id

}, dl\_cb, nil)

end

M.createSecretChat = createSecretChat

-- Creates new group chat and send corresponding messageGroupChatCreate, returns created chat @user\_ids Identifiers of users to add to the group @title Title of new group chat, 0-255 characters

local function createNewGroupChat(user\_ids, title)

tdcli\_function ({

ID = "CreateNewGroupChat",

user\_ids\_ = user\_ids, -- vector

title\_ = title

}, dl\_cb, nil)

end

M.createNewGroupChat = createNewGroupChat

-- Creates new channel chat and send corresponding messageChannelChatCreate, returns created chat @title Title of new channel chat, 0-255 characters @is\_supergroup True, if supergroup chat should be created @about Information about the channel, 0-255 characters

local function createNewChannelChat(title, is\_supergroup, about)

tdcli\_function ({

ID = "CreateNewChannelChat",

title\_ = title,

is\_supergroup\_ = is\_supergroup,

about\_ = about

}, dl\_cb, nil)

end

M.createNewChannelChat = createNewChannelChat

-- CRASHED

-- Creates new secret chat, returns created chat @user\_id Identifier of a user to create secret chat with

local function createNewSecretChat(user\_id)

tdcli\_function ({

ID = "CreateNewSecretChat",

user\_id\_ = user\_id

}, dl\_cb, nil)

end

M.createNewSecretChat = createNewSecretChat

-- Creates new channel supergroup chat from existing group chat and send corresponding messageChatMigrateTo and messageChatMigrateFrom. Deactivates group @chat\_id Group chat identifier

local function migrateGroupChatToChannelChat(chat\_id)

tdcli\_function ({

ID = "MigrateGroupChatToChannelChat",

chat\_id\_ = chat\_id

}, dl\_cb, nil)

end

M.migrateGroupChatToChannelChat = migrateGroupChatToChannelChat

-- Changes chat title. Title can't be changed for private chats. Title will not change until change will be synchronized with the server. Title will not be changed if application is killed before it can send request to the server.

-- - There will be update about change of the title on success. Otherwise error will be returned

-- @chat\_id Chat identifier @title New title of a chat, 0-255 characters

local function changeChatTitle(chat\_id, title)

tdcli\_function ({

ID = "ChangeChatTitle",

chat\_id\_ = chat\_id,

title\_ = title

}, dl\_cb, nil)

end

M.changeChatTitle = changeChatTitle

-- Changes chat photo. Photo can't be changed for private chats. Photo will not change until change will be synchronized with the server. Photo will not be changed if application is killed before it can send request to the server.

-- - There will be update about change of the photo on success. Otherwise error will be returned @chat\_id Chat identifier @photo New chat photo. You can use zero InputFileId to delete photo. Files accessible only by HTTP URL are not acceptable

local function changeChatPhoto(chat\_id, file)

tdcli\_function ({

ID = "ChangeChatPhoto",

chat\_id\_ = chat\_id,

photo\_ = {

ID = "InputFileLocal",

path\_ = file

}

}, dl\_cb, nil)

end

M.changeChatPhoto = changeChatPhoto

-- Changes chat draft message @chat\_id Chat identifier @draft\_message New draft message, nullable

local function changeChatDraftMessage(chat\_id, reply\_to\_message\_id, text, disable\_web\_page\_preview, clear\_draft, parse\_mode)

local TextParseMode = getParseMode(parse\_mode)

tdcli\_function ({

ID = "ChangeChatDraftMessage",

chat\_id\_ = chat\_id,

draft\_message\_ = {

ID = "DraftMessage",

reply\_to\_message\_id\_ = reply\_to\_message\_id,

input\_message\_text\_ = {

ID = "InputMessageText",

text\_ = text,

disable\_web\_page\_preview\_ = disable\_web\_page\_preview,

clear\_draft\_ = clear\_draft,

entities\_ = {},

parse\_mode\_ = TextParseMode,

},

},

}, dl\_cb, nil)

end

M.changeChatDraftMessage = changeChatDraftMessage

-- Adds new member to chat. Members can't be added to private or secret chats. Member will not be added until chat state will be synchronized with the server. Member will not be added if application is killed before it can send request to the server

-- @chat\_id Chat identifier @user\_id Identifier of the user to add @forward\_limit Number of previous messages from chat to forward to new member, ignored for channel chats

local function addChatMember(chat\_id, user\_id, forward\_limit)

tdcli\_function ({

ID = "AddChatMember",

chat\_id\_ = chat\_id,

user\_id\_ = user\_id,

forward\_limit\_ = forward\_limit

}, dl\_cb, nil)

end

M.addChatMember = addChatMember

-- Adds many new members to the chat. Currently, available only for channels. Can't be used to join the channel. Member will not be added until chat state will be synchronized with the server. Member will not be added if application is killed before it can send request to the server

-- @chat\_id Chat identifier @user\_ids Identifiers of the users to add

local function addChatMembers(chat\_id, user\_ids)

tdcli\_function ({

ID = "AddChatMembers",

chat\_id\_ = chat\_id,

user\_ids\_ = user\_ids -- vector

}, dl\_cb, nil)

end

M.addChatMembers = addChatMembers

-- Changes status of the chat member, need appropriate privileges. In channel chats, user will be added to chat members if he is yet not a member and there is less than 200 members in the channel.

-- Status will not be changed until chat state will be synchronized with the server. Status will not be changed if application is killed before it can send request to the server

-- @chat\_id Chat identifier @user\_id Identifier of the user to edit status, bots can be editors in the channel chats @status New status of the member in the chat

-- status = Creator|Editor|Moderator|Member|Left|Kicked

local function changeChatMemberStatus(chat\_id, user\_id, status)

tdcli\_function ({

ID = "ChangeChatMemberStatus",

chat\_id\_ = chat\_id,

user\_id\_ = user\_id,

status\_ = {

ID = "ChatMemberStatus" .. status

},

}, dl\_cb, nil)

end

M.changeChatMemberStatus = changeChatMemberStatus

-- Returns information about one participant of the chat @chat\_id Chat identifier @user\_id User identifier

local function getChatMember(chat\_id, user\_id)

tdcli\_function ({

ID = "GetChatMember",

chat\_id\_ = chat\_id,

user\_id\_ = user\_id

}, dl\_cb, nil)

end

M.getChatMember = getChatMember

-- Asynchronously downloads file from cloud. Updates updateFileProgress will notify about download progress. Update updateFile will notify about successful download @file\_id Identifier of file to download

local function downloadFile(file\_id)

tdcli\_function ({

ID = "DownloadFile",

file\_id\_ = file\_id

}, dl\_cb, nil)

end

M.downloadFile = downloadFile

-- Stops file downloading. If file already downloaded do nothing. @file\_id Identifier of file to cancel download

local function cancelDownloadFile(file\_id)

tdcli\_function ({

ID = "CancelDownloadFile",

file\_id\_ = file\_id

}, dl\_cb, nil)

end

M.cancelDownloadFile = cancelDownloadFile

-- Generates new chat invite link, previously generated link is revoked. Available for group and channel chats. Only creator of the chat can export chat invite link @chat\_id Chat identifier

local function exportChatInviteLink(chat\_id)

tdcli\_function ({

ID = "ExportChatInviteLink",

chat\_id\_ = chat\_id

}, dl\_cb, nil)

end

M.exportChatInviteLink = exportChatInviteLink

-- Checks chat invite link for validness and returns information about the corresponding chat @invite\_link Invite link to check. Should begin with "https:-- telegram.me/joinchat/"

local function checkChatInviteLink(link,cb)

tdcli\_function ({

ID = "CheckChatInviteLink",

invite\_link\_ = link

}, cb, nil)

end

M.checkChatInviteLink = checkChatInviteLink

-- Imports chat invite link, adds current user to a chat if possible. Member will not be added until chat state will be synchronized with the server. Member will not be added if application is killed before it can send request to the server

-- @invite\_link Invite link to import. Should begin with "https:-- telegram.me/joinchat/"

local function importChatInviteLink(invite\_link)

tdcli\_function ({

ID = "ImportChatInviteLink",

invite\_link\_ = invite\_link

}, dl\_cb, nil)

end

M.importChatInviteLink = importChatInviteLink

-- Adds user to black list @user\_id User identifier

local function blockUser(user\_id)

tdcli\_function ({

ID = "BlockUser",

user\_id\_ = user\_id

}, dl\_cb, nil)

end

M.blockUser = blockUser

-- Removes user from black list @user\_id User identifier

local function unblockUser(user\_id)

tdcli\_function ({

ID = "UnblockUser",

user\_id\_ = user\_id

}, dl\_cb, nil)

end

M.unblockUser = unblockUser

-- Returns users blocked by the current user @offset Number of users to skip in result, must be non-negative @limit Maximum number of users to return, can't be greater than 100

local function getBlockedUsers(offset, limit,cb)

tdcli\_function ({

ID = "GetBlockedUsers",

offset\_ = offset,

limit\_ = limit

}, dl\_cb, nil)

end

M.getBlockedUsers = getBlockedUsers

-- Adds new contacts/edits existing contacts, contacts user identifiers are ignored. Returns list of corresponding users in the same order as input contacts. If contact doesn't registered in Telegram, user with id == 0 will be returned @contacts List of contacts to import/edit

local function importContacts(phone\_number, first\_name, last\_name, user\_id)

tdcli\_function ({

ID = "ImportContacts",

contacts\_ = {[0] = {

phone\_number\_ = tostring(phone\_number),

first\_name\_ = tostring(first\_name),

last\_name\_ = tostring(last\_name),

user\_id\_ = user\_id

},

},

}, dl\_cb, nil)

end

M.importContacts = importContacts

-- Searches for specified query in the first name, last name and username of the known user contacts @query Query to search for, can be empty to return all contacts @limit Maximum number of users to be returned

local function searchContacts(query, limit)

tdcli\_function ({

ID = "SearchContacts",

query\_ = query,

limit\_ = limit

}, dl\_cb, nil)

end

M.searchContacts = searchContacts

-- Deletes users from contacts list @user\_ids Identifiers of users to be deleted

local function deleteContacts(user\_ids)

tdcli\_function ({

ID = "DeleteContacts",

user\_ids\_ = user\_ids -- vector

}, dl\_cb, nil)

end

M.deleteContacts = deleteContacts

-- Returns profile photos of the user. Result of this query can't be invalidated, so it must be used with care @user\_id User identifier @offset Photos to skip, must be non-negative @limit Maximum number of photos to be returned, can't be greater than 100

local function getUserProfilePhotos(user\_id, offset, limit,cb)

tdcli\_function ({

ID = "GetUserProfilePhotos",

user\_id\_ = user\_id,

offset\_ = offset,

limit\_ = limit

}, cb, nil)

end

M.getUserProfilePhotos = getUserProfilePhotos

-- Returns stickers corresponding to given emoji @emoji String representation of emoji. If empty, returns all known stickers

local function getStickers(emoji,cb)

tdcli\_function ({

ID = "GetStickers",

emoji\_ = emoji

}, cb, nil)

end

M.getStickers = getStickers

-- Returns list of installed sticker sets @only\_enabled If true, returns only enabled sticker sets

local function getStickerSets(only\_enabled)

tdcli\_function ({

ID = "GetStickerSets",

only\_enabled\_ = only\_enabled

}, dl\_cb, nil)

end

M.getStickerSets = getStickerSets

-- Returns information about sticker set by its identifier @set\_id Identifier of the sticker set

local function getStickerSet(set\_id)

tdcli\_function ({

ID = "GetStickerSet",

set\_id\_ = set\_id

}, dl\_cb, nil)

end

M.getStickerSet = getStickerSet

-- Searches sticker set by its short name @name Name of the sticker set

local function searchStickerSet(name)

tdcli\_function ({

ID = "SearchStickerSet",

name\_ = name

}, dl\_cb, nil)

end

M.searchStickerSet = searchStickerSet

-- Installs/uninstalls or enables/archives sticker set. Official sticker set can't be uninstalled, but it can be archived @set\_id Identifier of the sticker set @is\_installed New value of is\_installed @is\_enabled New value of is\_enabled

local function updateStickerSet(set\_id, is\_installed, is\_enabled)

tdcli\_function ({

ID = "UpdateStickerSet",

set\_id\_ = set\_id,

is\_installed\_ = is\_installed,

is\_enabled\_ = is\_enabled

}, dl\_cb, nil)

end

M.updateStickerSet = updateStickerSet

-- Returns saved animations

local function getSavedAnimations()

tdcli\_function ({

ID = "GetSavedAnimations",

}, dl\_cb, nil)

end

M.getSavedAnimations = getSavedAnimations

-- Manually adds new animation to the list of saved animations. New animation is added to the beginning of the list. If the animation is already in the list, at first it is removed from the list. Only video animations with MIME type "video/mp4" can be added to the list

-- @animation Animation file to add. Only known to server animations (i. e. successfully sent via message) can be added to the list

local function addSavedAnimation(id)

tdcli\_function ({

ID = "AddSavedAnimation",

animation\_ = {

ID = "InputFileId",

id\_ = id

},

}, dl\_cb, nil)

end

M.addSavedAnimation = addSavedAnimation

-- Removes animation from the list of saved animations @animation Animation file to delete

local function deleteSavedAnimation(id)

tdcli\_function ({

ID = "DeleteSavedAnimation",

animation\_ = {

ID = "InputFileId",

id\_ = id

},

}, dl\_cb, nil)

end

M.deleteSavedAnimation = deleteSavedAnimation

-- Returns up to 20 recently used inline bots in the order of the last usage

local function getRecentInlineBots()

tdcli\_function ({

ID = "GetRecentInlineBots",

}, dl\_cb, nil)

end

M.getRecentInlineBots = getRecentInlineBots

-- Get web page preview by text of the message. Do not call this function to often @message\_text Message text

local function getWebPagePreview(message\_text)

tdcli\_function ({

ID = "GetWebPagePreview",

message\_text\_ = message\_text

}, dl\_cb, nil)

end

M.getWebPagePreview = getWebPagePreview

-- Returns notification settings for given scope @scope Scope to return information about notification settings

-- scope = Chat(chat\_id)|PrivateChats|GroupChats|AllChats|

local function getNotificationSettings(scope, chat\_id)

tdcli\_function ({

ID = "GetNotificationSettings",

scope\_ = {

ID = 'NotificationSettingsFor' .. scope,

chat\_id\_ = chat\_id or nil

},

}, dl\_cb, nil)

end

M.getNotificationSettings = getNotificationSettings

-- Changes notification settings for given scope @scope Scope to change notification settings

-- @notification\_settings New notification settings for given scope

-- scope = Chat(chat\_id)|PrivateChats|GroupChats|AllChats|

local function setNotificationSettings(scope, chat\_id, mute\_for, show\_preview)

tdcli\_function ({

ID = "SetNotificationSettings",

scope\_ = {

ID = 'NotificationSettingsFor' .. scope,

chat\_id\_ = chat\_id or nil

},

notification\_settings\_ = {

ID = "NotificationSettings",

mute\_for\_ = mute\_for,

sound\_ = "default",

show\_preview\_ = show\_preview

}

}, dl\_cb, nil)

end

M.setNotificationSettings = setNotificationSettings

-- Uploads new profile photo for logged in user. Photo will not change until change will be synchronized with the server. Photo will not be changed if application is killed before it can send request to the server. If something changes, updateUser will be sent @photo\_path Path to new profile photo

local function setProfilePhoto(photo\_path)

tdcli\_function ({

ID = "SetProfilePhoto",

photo\_path\_ = photo\_path

}, dl\_cb, nil)

end

M.setProfilePhoto = setProfilePhoto

-- Deletes profile photo. If something changes, updateUser will be sent @profile\_photo\_id Identifier of profile photo to delete

local function deleteProfilePhoto(profile\_photo\_id)

tdcli\_function ({

ID = "DeleteProfilePhoto",

profile\_photo\_id\_ = profile\_photo\_id

}, dl\_cb, nil)

end

M.deleteProfilePhoto = deleteProfilePhoto

-- Changes first and last names of logged in user. If something changes, updateUser will be sent @first\_name New value of user first name, 1-255 characters @last\_name New value of optional user last name, 0-255 characters

local function changeName(first\_name, last\_name)

tdcli\_function ({

ID = "ChangeName",

first\_name\_ = first\_name,

last\_name\_ = last\_name

}, dl\_cb, nil)

end

M.changeName = changeName

-- Changes about information of logged in user @about New value of userFull.about, 0-255 characters

local function changeAbout(about)

tdcli\_function ({

ID = "ChangeAbout",

about\_ = about

}, dl\_cb, nil)

end

M.changeAbout = changeAbout

-- Changes username of logged in user. If something changes, updateUser will be sent @username New value of username. Use empty string to remove username

local function changeUsername(username)

tdcli\_function ({

ID = "ChangeUsername",

username\_ = username

}, dl\_cb, nil)

end

M.changeUsername = changeUsername

-- Changes user's phone number and sends authentication code to the new user's phone number. Returns authStateWaitCode with information about sent code on success

-- @phone\_number New user's phone number in any reasonable format @allow\_flash\_call Pass True, if code can be sent via flash call to the specified phone number @is\_current\_phone\_number Pass true, if the phone number is used on the current device. Ignored if allow\_flash\_call is False

local function changePhoneNumber(phone\_number, allow\_flash\_call, is\_current\_phone\_number)

tdcli\_function ({

ID = "ChangePhoneNumber",

phone\_number\_ = phone\_number,

allow\_flash\_call\_ = allow\_flash\_call,

is\_current\_phone\_number\_ = is\_current\_phone\_number

}, dl\_cb, nil)

end

M.changePhoneNumber = changePhoneNumber

-- Resends authentication code sent to change user's phone number. Wotks only if in previously received authStateWaitCode next\_code\_type was not null. Returns authStateWaitCode on success

local function resendChangePhoneNumberCode()

tdcli\_function ({

ID = "ResendChangePhoneNumberCode",

}, dl\_cb, nil)

end

M.resendChangePhoneNumberCode = resendChangePhoneNumberCode

-- Checks authentication code sent to change user's phone number. Returns authStateOk on success @code Verification code from SMS, voice call or flash call

local function checkChangePhoneNumberCode(code)

tdcli\_function ({

ID = "CheckChangePhoneNumberCode",

code\_ = code

}, dl\_cb, nil)

end

M.checkChangePhoneNumberCode = checkChangePhoneNumberCode

-- Returns all active sessions of logged in user

local function getActiveSessions()

tdcli\_function ({

ID = "GetActiveSessions",

}, dl\_cb, nil)

end

M.getActiveSessions = getActiveSessions

-- Terminates another session of logged in user @session\_id Session identifier

local function terminateSession(session\_id)

tdcli\_function ({

ID = "TerminateSession",

session\_id\_ = session\_id

}, dl\_cb, nil)

end

M.terminateSession = terminateSession

-- Terminates all other sessions of logged in user

local function terminateAllOtherSessions()

tdcli\_function ({

ID = "TerminateAllOtherSessions",

}, dl\_cb, nil)

end

M.terminateAllOtherSessions = terminateAllOtherSessions

-- Gives or revokes all members of the group editor rights. Needs creator privileges in the group @group\_id Identifier of the group @anyone\_can\_edit New value of anyone\_can\_edit

local function toggleGroupEditors(group\_id, anyone\_can\_edit)

tdcli\_function ({

ID = "ToggleGroupEditors",

group\_id\_ = getChatId(group\_id).ID,

anyone\_can\_edit\_ = anyone\_can\_edit

}, dl\_cb, nil)

end

M.toggleGroupEditors = toggleGroupEditors

-- Changes username of the channel. Needs creator privileges in the channel @channel\_id Identifier of the channel @username New value of username. Use empty string to remove username

local function changeChannelUsername(channel\_id, username)

tdcli\_function ({

ID = "ChangeChannelUsername",

channel\_id\_ = getChatId(channel\_id).ID,

username\_ = username

}, dl\_cb, nil)

end

M.changeChannelUsername = changeChannelUsername

-- Gives or revokes right to invite new members to all current members of the channel. Needs creator privileges in the channel. Available only for supergroups @channel\_id Identifier of the channel @anyone\_can\_invite New value of anyone\_can\_invite

local function toggleChannelInvites(channel\_id, anyone\_can\_invite)

tdcli\_function ({

ID = "ToggleChannelInvites",

channel\_id\_ = getChatId(channel\_id).ID,

anyone\_can\_invite\_ = anyone\_can\_invite

}, dl\_cb, nil)

end

M.toggleChannelInvites = toggleChannelInvites

-- Enables or disables sender signature on sent messages in the channel. Needs creator privileges in the channel. Not available for supergroups @channel\_id Identifier of the channel @sign\_messages New value of sign\_messages

local function toggleChannelSignMessages(channel\_id, sign\_messages)

tdcli\_function ({

ID = "ToggleChannelSignMessages",

channel\_id\_ = getChatId(channel\_id).ID,

sign\_messages\_ = sign\_messages

}, dl\_cb, nil)

end

M.toggleChannelSignMessages = toggleChannelSignMessages

-- Changes information about the channel. Needs creator privileges in the broadcast channel or editor privileges in the supergroup channel @channel\_id Identifier of the channel @about New value of about, 0-255 characters

local function changeChannelAbout(channel\_id, about)

tdcli\_function ({

ID = "ChangeChannelAbout",

channel\_id\_ = getChatId(channel\_id).ID,

about\_ = about

}, dl\_cb, nil)

end

M.changeChannelAbout = changeChannelAbout

-- Pins a message in a supergroup channel chat. Needs editor privileges in the channel @channel\_id Identifier of the channel @message\_id Identifier of the new pinned message @disable\_notification True, if there should be no notification about the pinned message

local function pinChannelMessage(channel\_id, message\_id,disable\_notification)

tdcli\_function ({

ID = "PinChannelMessage",

channel\_id\_ = getChatId(channel\_id).ID,

message\_id\_ = message\_id,

disable\_notification\_ = disable\_notification,

}, dl\_cb, nil)

end

M.pinChannelMessage = pinChannelMessage

-- Removes pinned message in the supergroup channel. Needs editor privileges in the channel @channel\_id Identifier of the channel

local function unpinChannelMessage(channel\_id)

tdcli\_function ({

ID = "UnpinChannelMessage",

channel\_id\_ = getChatId(channel\_id).ID

}, dl\_cb, nil)

end

M.unpinChannelMessage = unpinChannelMessage

-- Reports some supergroup channel messages from a user as spam messages @channel\_id Channel identifier @user\_id User identifier @message\_ids Identifiers of messages sent in the supergroup by the user, the list should be non-empty

local function reportChannelSpam(channel\_id, user\_id, message\_ids)

tdcli\_function ({

ID = "ReportChannelSpam",

channel\_id\_ = getChatId(channel\_id).ID,

user\_id\_ = user\_id,

message\_ids\_ = message\_ids -- vector

}, dl\_cb, nil)

end

M.reportChannelSpam = reportChannelSpam

-- Returns information about channel members or kicked from channel users. Can be used only if channel\_full->can\_get\_members == true @channel\_id Identifier of the channel

-- @filter Kind of channel users to return, defaults to channelMembersRecent @offset Number of channel users to skip @limit Maximum number of users be returned, can't be greater than 200

-- filter = Recent|Administrators|Kicked|Bots

local function getChannelMembers(channel\_id, offset, filter, limit,cb)

tdcli\_function ({

ID = "GetChannelMembers",

channel\_id\_ = getChatId(channel\_id).ID,

filter\_ = {

ID = "ChannelMembers" .. filter

},

offset\_ = offset,

limit\_ = limit

}, cb, nil)

end

M.getChannelMembers = getChannelMembers

-- Deletes channel along with all messages in corresponding chat. Releases channel username and removes all members. Needs creator privileges in the channel. Channels with more than 1000 members can't be deleted @channel\_id Identifier of the channel

local function deleteChannel(channel\_id)

tdcli\_function ({

ID = "DeleteChannel",

channel\_id\_ = getChatId(channel\_id).ID

}, dl\_cb, nil)

end

M.deleteChannel = deleteChannel

-- Returns user that can be contacted to get support

local function getSupportUser(cb)

tdcli\_function ({

ID = "GetSupportUser",

}, cb, nil)

end

M.getSupportUser = getSupportUser

-- Returns background wallpapers

local function getWallpapers(cb)

tdcli\_function ({

ID = "GetWallpapers",

}, cb, nil)

end

M.getWallpapers = getWallpapers

local function registerDevice(cb)

tdcli\_function ({

ID = "RegisterDevice",

}, cb, nil)

end

M.registerDevice = registerDevice

--registerDevice device\_token:DeviceToken = Ok;

local function getDeviceTokens()

tdcli\_function ({

ID = "GetDeviceTokens",

}, dl\_cb, nil)

end

M.getDeviceTokens = getDeviceTokens

-- CRASHED

-- Changes privacy settings @key Privacy key @rules New privacy rules

-- key = UserStatus|ChatInvite

-- rules = AllowAll|AllowContacts|AllowUsers(user\_ids)|DisallowAll|DisallowContacts|DisallowUsers(user\_ids)

local function setPrivacy(key, rules, user\_ids)

if user\_ids and rules:match('Allow') then

rule = 'AllowUsers'

elseif user\_ids and rules:match('Disallow') then

rule = 'DisallowUsers'

end

tdcli\_function ({

ID = "SetPrivacy",

key\_ = {

ID = 'PrivacyKey' .. key,

},

rules\_ = {

ID = 'PrivacyRules',

rules\_ = {

[0] = {

ID = 'PrivacyRule' .. rules,

},

{

ID = 'PrivacyRule' .. rule,

user\_ids\_ = user\_ids

},

},

},

}, dl\_cb, nil)

end

M.setPrivacy = setPrivacy

-- Returns current privacy settings @key Privacy key

-- key = UserStatus|ChatInvite

local function getPrivacy(key)

tdcli\_function ({

ID = "GetPrivacy",

key\_ = {

ID = "PrivacyKey" .. key

},

}, dl\_cb, nil)

end

M.getPrivacy = getPrivacy

-- Returns value of an option by its name. See list of available options on https://core.telegram.org/tdlib/options

-- @name Name of the option

local function getOption(name)

tdcli\_function ({

ID = "GetOption",

name\_ = name

}, dl\_cb, nil)

end

M.getOption = getOption

-- CRASHED

-- Sets value of an option. See list of available options on https://core.telegram.org/tdlib/options. Only writable options can be set

-- @name Name of the option @value New value of the option

local function setOption(name, option, value)

tdcli\_function ({

ID = "SetOption",

name\_ = name,

value\_ = {

ID = 'Option' .. option,

value\_ = value

},

}, dl\_cb, nil)

end

M.setOption = setOption

-- Changes period of inactivity, after which the account of currently logged in user will be automatically deleted @ttl New account TTL

local function changeAccountTtl(days)

tdcli\_function ({

ID = "ChangeAccountTtl",

ttl\_ = {

ID = "AccountTtl",

days\_ = days

},

}, dl\_cb, nil)

end

M.changeAccountTtl = changeAccountTtl

-- Returns period of inactivity, after which the account of currently logged in user will be automatically deleted

local function getAccountTtl()

tdcli\_function ({

ID = "GetAccountTtl",

}, dl\_cb, nil)

end

M.getAccountTtl = getAccountTtl

-- Deletes the account of currently logged in user, deleting from the server all information associated with it. Account's phone number can be used to create new account, but only once in two weeks @reason Optional reason of account deletion

local function deleteAccount(reason)

tdcli\_function ({

ID = "DeleteAccount",

reason\_ = reason

}, dl\_cb, nil)

end

M.deleteAccount = deleteAccount

-- Returns current chat report spam state @chat\_id Chat identifier

local function getChatReportSpamState(chat\_id)

tdcli\_function ({

ID = "GetChatReportSpamState",

chat\_id\_ = chat\_id

}, dl\_cb, nil)

end

M.getChatReportSpamState = getChatReportSpamState

-- Reports chat as a spam chat or as not a spam chat. Can be used only if ChatReportSpamState.can\_report\_spam is true. After this request ChatReportSpamState.can\_report\_spam became false forever @chat\_id Chat identifier @is\_spam\_chat If true, chat will be reported as a spam chat, otherwise it will be marked as not a spam chat

local function changeChatReportSpamState(chat\_id, is\_spam\_chat)

tdcli\_function ({

ID = "ChangeChatReportSpamState",

chat\_id\_ = chat\_id,

is\_spam\_chat\_ = is\_spam\_chat

}, dl\_cb, nil)

end

M.changeChatReportSpamState = changeChatReportSpamState

-- Bots only. Informs server about number of pending bot updates if they aren't processed for a long time @pending\_update\_count Number of pending updates @error\_message Last error's message

local function setBotUpdatesStatus(pending\_update\_count, error\_message)

tdcli\_function ({

ID = "SetBotUpdatesStatus",

pending\_update\_count\_ = pending\_update\_count,

error\_message\_ = error\_message

}, dl\_cb, nil)

end

M.setBotUpdatesStatus = setBotUpdatesStatus

-- Returns Ok after specified amount of the time passed @seconds Number of seconds before that function returns

local function setAlarm(seconds)

tdcli\_function ({

ID = "SetAlarm",

seconds\_ = seconds

}, dl\_cb, nil)

end

M.setAlarm = setAlarm

-- These functions below are an effort to mimic telegram-cli console commands --

-- Tries to add user to contact list

local function add\_contact(phone, first\_name, last\_name, user\_id)

bot.importContacts(phone, first\_name, last\_name, user\_id)

end

M.add\_contact = add\_contact

-- Gets channel admins

local function channel\_get\_admins(channel,cb)

local function callback\_admins(extra,result,success)

limit = result.administrator\_count\_

if tonumber(limit) > 0 then

bot.getChannelMembers(channel, 0, 'Administrators', limit,cb)

else return bot.sendMessage(channel, 0, 1,'ربات ادمین گروه نشده است !', 1, 'html') end

end

bot.getChannelFull(channel,callback\_admins)

end

M.channel\_get\_admins = channel\_get\_admins

-- Gets channel bot.

local function channel\_get\_bots(channel,cb)

local function callback\_admins(extra,result,success)

limit = result.member\_count\_

bot.getChannelMembers(channel, 0, 'Bots', limit,cb)

end

bot.getChannelFull(channel,callback\_admins)

end

M.channel\_get\_bots = channel\_get\_bots

-- Gets channel kicked members

local function channel\_get\_kicked(channel,cb)

local function callback\_admins(extra,result,success)

limit = result.kicked\_count\_

bot.getChannelMembers(channel, 0, 'Kicked', limit,cb)

end

bot.getChannelFull(channel,callback\_admins)

end

M.channel\_get\_kicked = channel\_get\_kicked

-- changes value of basic channel parameters.

-- param=sign|invites

local function channel\_edit(channel\_id, param, enabled)

local channel\_id = getChatId(channel\_id).ID

if param:lower() == 'sign' then

bot.toggleChannelSignMessages(channel\_id, enabled)

elseif param:lower() == 'invites' then

bot.toggleChannelInvites(channel\_id, enabled)

end

end

M.channel\_edit = channel\_edit

-- changes user's role in chat.

-- role=Creator|Editor|Moderator|Member|Left|Kicked

local function chat\_change\_role(chat\_id, user\_id, role)

bot.changeChatMemberStatus(chat\_id, user\_id, role)

end

M.chat\_change\_role = chat\_change\_role

-- Deletes user from chat

local function chat\_del\_user(chat\_id, user\_id)

bot.changeChatMemberStatus(chat\_id, user\_id, 'Editor')

end

M.chat\_del\_user = chat\_del\_user

-- Prints info about chat

local function chat\_info(chat\_id)

bot.getChat(chat\_id)

end

M.chat\_info = chat\_info

-- Joins to chat (by invite link)

local function chat\_join(chat\_id)

bot.importChatInviteLink(chat\_id)

end

M.chat\_join = chat\_join

-- Leaves chat

local function chat\_leave(chat\_id, user\_id)

bot.changeChatMemberStatus(chat\_id, user\_id, "Left")

end

M.chat\_leave = chat\_leave

-- Creates broadcast channel

local function chat\_create\_broadcast(title, about)

bot.createNewChannelChat(title, 0, about)

end

M.chat\_create\_broadcast = chat\_create\_broadcast

-- Creates group chat

local function chat\_create\_group(title, user\_ids)

bot.createNewGroupChat(title, user\_ids)

end

M.chat\_create\_group = chat\_create\_group

-- Creates supergroup channel

local function chat\_create\_supergroup(title, about)

bot.createNewChannelChat(title, 1, about)

end

M.chat\_create\_supergroup = chat\_create\_supergroup

-- Prints contact list

local function contact\_list(limit)

bot.searchContacts("", limit)

end

M.contact\_list = contact\_list

-- List of last conversations

local function dialog\_list(limit)

bot.searchChats("", limit)

end

M.dialog\_list = dialog\_list

-- Upgrades group to supergroup

local function group\_upgrade(chat\_id)

bot.migrateGroupChatToChannelChat(chat\_id)

end

M.group\_upgrade = group\_upgrade

-- Marks messages with peer as read

local function mark\_read(chat\_id, message\_ids)

bot.viewMessages(chat\_id, message\_ids)

end

M.mark\_read = mark\_read

-- mutes chat for specified number of seconds (default 60)

local function mute(chat\_id, mute\_for)

bot.setNotificationSettings(chat\_id, mute\_for or 60, 0)

end

M.mute = mute

-- Tries to push inline button

local function push\_button(message, button\_id)

end

M.push\_button = push\_button

-- Find chat by username

local function resolve\_username(username,cb)

tdcli\_function ({

ID = "SearchPublicChat",

username\_ = username

}, cb, nil)

end

M.resolve\_username = resolve\_username

-- Replies to peer with file

local function reply\_file(chat\_id, msg\_id, type, file, caption)

tdcli\_function ({

ID = "SendMessage",

chat\_id\_ = chat\_id,

reply\_to\_message\_id\_ = msg\_id,

disable\_notification\_ = 0,

from\_background\_ = 1,

reply\_markup\_ = nil,

input\_message\_content\_ = getInputMessageContent(file, type, caption),

}, dl\_cb, nil)

end

M.reply\_file = reply\_file

-- Forwards message to peer. Forward to secret chats is forbidden

local function reply\_fwd(msg\_id, fwd\_id)

end

M.reply\_fwd = reply\_fwd

-- Sends geo location

local function reply\_location(chat\_id, msg\_id, latitude, longitude)

tdcli\_function ({

ID="SendMessage",

chat\_id\_=chat\_id,

reply\_to\_message\_id\_=msg\_id,

disable\_notification\_=0,

from\_background\_=1,

reply\_markup\_=nil,

input\_message\_content\_={

ID="InputMessageLocation",

location\_={

ID = "Location",

latitude\_ = latitude,

longitude\_ = longitude

},

},

}, dl\_cb, nil)

end

M.reply\_location = reply\_location

-- Sends file to peer

local function send\_file(chat\_id, type, file, caption,cb)

tdcli\_function ({

ID = "SendMessage",

chat\_id\_ = chat\_id,

reply\_to\_message\_id\_ = 0,

disable\_notification\_ = 0,

from\_background\_ = 1,

reply\_markup\_ = nil,

input\_message\_content\_ = getInputMessageContent(file, type, caption),

}, cb, nil)

end

M.send\_file = send\_file

-- Sends geo location

local function send\_location(chat\_id, latitude, longitude)

tdcli\_function ({

ID = "SendMessage",

chat\_id\_ = chat\_id,

reply\_to\_message\_id\_ = 0,

disable\_notification\_ = 0,

from\_background\_ = 1,

reply\_markup\_ = nil,

input\_message\_content\_ = {

ID = "InputMessageLocation",

location\_ = {

ID = "Location",

latitude\_ = latitude,

longitude\_ = longitude

},

},

}, dl\_cb, nil)

end

M.send\_location = send\_location

-- Adds bot to chat

local function start\_bot(user\_id, chat\_id, data)

bot.sendBotStartMessage(user\_id, chat\_id, 'start')

end

M.start\_bot = start\_bot

-- sets timer (in seconds)

local function timer(timeout)

bot.setAlarm(timeout)

end

M.timer = timer

-- unmutes chat

local function unmute(chat\_id)

bot.setNotificationSettings(chat\_id, 0, 1)

end

M.unmute = unmute

return M