SKYPE API

Description of Skype API and how to use it

Maintainer: Taavet Hinrikus
Version: 1.2 2005-03-04

1 PURPOSE OF THIS DOCUMENT

Describe the Skype API and how to implement applications and devices communicating with Skype through the API.

2 REVISION HISTORY

2.1API

Revision: 1.0.0.94
Date: 2004-10-21
Release notes: API public release

Revision: 1.1.0.61
Date: 2005-01-12
Release notes:

- added: Protocol 3
- change: API - now allows one ongoing search per client only. Attempting to issue new search before receiving results of previous one will result in error 72.
- change: API allows now one ongoing search per client only
- change: CHAT and CHATMESSAGE properties
- bugfix: API showed previous user's calls and messages
- bugfix: Fixed confusing syntax if protocol 3 is used
- bugfix: SEARCH MESSAGES does not return CHATMESSAGES value anymore if protocol 2 is used
- bugfix: API displayed only first word of message or fullname
- bugfix: In ACL only one program's permission was remembered
- bugfix: MC message IDs were not returned
- bugfix: Problems with connecting for older applications
- bugfix: Fixed API exceptions if Skype is used on two Windows accounts simultainously
- bugfix: On Win98/Me lots of dll files were shown to use Skype instead of the respective application
- bugfix: Sometimes API didn't return 'BUDDYSTATUS 1' messages

Revision: 1.2.0.11
Date: 2005-03-04
Release notes:

- added: Protocol 4
- Support for conferencing: start a conference, add people to conference and being able to get list of conf call participants and notifications about

these

- Possibility to check SkypeOut balance
- Possible to call speeddial numbers
- Notifications about changing audiodevices
- Notification about deleting IM history
- Changed language and country to return ISO list instead of countrynames (new behaviour: starting from protocol 4 language and country values will be prefixed by ISO codes ('GET USER echo123 COUNTRY' => 'USER echo123 COUNTRY ee Estonia'))
- Notification about shutting down Skype
- Support for SkypeIn
- Registry key to disable one second timeout for debugging
- Possibility to add userhande to OPEN ADDAFRIEND
- CALL FAILUREREASON 1 - documentation error, doc changed to say "Misc error"
- change: if CHATMESSAGE property is missing, command 'SET CHATMESSAGE id' gives the same error for both existing and nonexisting id
- change: PSTN_STATUS gives error string returned from gateway
- change: HASCALLEQUIPMENT always returns TRUE
- bugfix: #11648 API: "AUDIO IN" and "AUDIO OUT" commands don't read doubly byte driver names correctly
- bugfix: #11473 API: BTN_PRESSED E fails with error 71 invalid key
- bugfix: #11472 API: mute is communicated
- bugfix: #11468 API: Conference inititated to more than 4 participants
- bugfix: #11368 API: IMHISTORYCHANGED doesn't work
- bugfix: #11272 API: MUTE doesn't work
- bugfix: #11102 API: Cannot call SkypeOut contacts using speeddial
- bugfix: #11344 API: no response to empty CALL
- bugfix: #11204 SKYPE access api canot deny access to a device.
- change: #10567 up/down via phone api autoexpand contactlist groups
- bugfix: #11111 API: No notification if the user changes audio device

2.2 Document

Rev	Date	Changed by	Comment/Change
1.0	2004-10-22	Taavet Hinrikus	API public release

1.1	2005-01-12	Taavet Hinrikus	Release of Skype 1.1
1.2	2005-03-04	Taavet Hinrikus	Release of Skype 1.2, protocol 4, developer FAQ entries, etc

3 LEGAL STATUS

This API documentation and the Skype API are copyrighted property of Skype Technologies S.A. or affiliated companies. For usage restrictions please read the EULA available on the Skype website http://www.skype.com/go/eula

4 INTRODUCTION

The Skype API is divided into 2 separate parts. Skype Phone API and Skype Access API:

- **Skype Phone API** is an interface that Skype uses to access devices, including, but not limited to, USB phones. The device by itself does not have to be a hardware device, but generally it is. This API is controlled by Skype and the device-side of the API can be viewed as a driver. The controlling party is Skype. This API operates on abstract events level e.g. the green button is pressed, the handset is off-hook, the device should ring, etc. The Skype Phone API compatible drivers should install themselves during install so that Skype can know their presence. There can be a database of USB devices and related drivers making it possible for Skype to detect a new device and prompt for driver installation.
- **Skype Access API** is an interface Skype publishes to 3rd party applications to access Skype functionality, for example to place a call, to send a text message, to get Skype user profile, etc. In this API, the 3rd party application is the controlling side. Skype can optionally grant access to the Skype Access API on a per-application basis.

5 GENERAL API OVERVIEW

The API is built up on simple text messages that are sent back and forth between Skype and a device (or device driver or controller running on the host computer).

6 PHONE API

Phone API commands that are implented as of now are listed below.

6.1 From Device to Skype

- NAME deviceName
- PROTOCOL version
- AUDIO_IN deviceName
- AUDIO_OUT deviceName
- HOOK ON|OFF

- MUTE ON|OFF
- BTN_PRESSED (0-9,A-Z,#,*,UP,DOWN,YES,NO,SKYPE)
- BTN_RELEASED ...

6.2 From Skype to Device

- MUTE ON|OFF

7 ACCESS API

Currently there are some commands (PROTOCOL, AUDIO_*) which are used in both the Skype Access API and the Skype Phone API.

7.1 Usage introduction

When a client application starts using Skype through the Skype API, then Skype will switch audio devices to the devices reported by the client through the API. When the transport layer connection dies or is dropped, then Skype will switch audio devices back to previously selected devices (Skype will regularly check that the transport layer is still alive). Possibly in the future a notification will be provided through the API when the user manually switched audio devices to something else (when compared to devices reported by active client).

It is possible that end-user confirmation is required for allowing a 3rd party to use Skype using the API. All actions performed using the API will be mirrored on the Skype application running on the computer.

Note that all times and dates in API are in UTC (Coordinated Universal Time).

7.2 Multiple Client Support

It is possible for multiple applications to use the Skype API at the same time. Protocol stays the same except the following for connecting to Skype: To initiate communication, Client should broadcast windows message ('SkypeControlAPIDiscover') to all windows in the system, specifying its own window handle in wParam parameter. In response, Skype responds with message 'SkypeControlAPIAttach' to the handle specified and indicates communication window handle in wParam.

Note that polling Skype with 'SkypeControlAPI' should not be done anymore, instead we will introduce a command for pinging in the next version.

7.3 API Access Control (ACL)

Whenever an application tries to use the Skype API, dialog will pop up, asking for user's confirmation - whether to allow the application to use the API or not. The main reason for this feature is to protect the privacy and security of the user.

Some suggestions for developers to keep things simple for users:

- Program executables (.exe file) should be named nicely e.g. "SkypeForWindows.exe". This is important, as the user will see the executable file name. If filename is unclear, then the user might decide

not allow this application to access Skype.

- Signing - applications should be signed with VeriSign's CodesSigning certificate.
- NAME command - application should support the "NAME" command and should publish its name

7.4SkypeOut

7.4.1 SkypeOut contacts

It is possible to get the list of SkypeOut contacts - these are now part of the main contact list i.e. they will be returned with the contact list numbers, if "SEARCH FRIENDS" command is executed.

To get more information about the number in current user's SkypeOut contacts - "GET USER <number> <fullname>".

For SkypeOut contacts, command "GET USER <number> ONLINESTATUS" will return "SKYPEOUT".

7.5How to detect Skype

To see if Skype is installed check the following registry key: *HKCU\Software\Skype\Phone '?SkypePath'* . The key will point to the location of skype.exe. In case that key is missing the application should also check for existance of *HKLM\Software\Skype\Phone '?SkypePath'* (if HKCU is missing, but HKLM is present, it indicates that skype has been installed from an administrator account, but not yet used from current account).

7.6Versioning info

Whenever a new version of the API is released the protocol version number is increased. When a client starts using the API then the client must tell the Skype API the latest protocol version that it supports. Skype will reply with its latest version number and the number reported by Skype will be the protocol version used. Skype will never reply with a protocol version which is newer than the version the client application supports. Skype defaults to protocol version 1. Skype-supported version can be queried with PROTOCOL 99999.

Example: Client speaks version 3 and tells Skype "PROTOCOL 3", Skype knows version 2 and replies with "PROTOCOL 2". Version 2 will be the used protocol in this case.

Notes about Skype API version changes and situations, where client supports an older version compared to Skype:

- Skype will not discard messages from newer protocol versions, but will perform the action requested in the message
- Client should ignore unknown commands and properties (which can be from a newer version of API, for example new status properties etc.)

7.7 Skype API protocol versions

Protocol versions 1,2 and 3 are available at the moment.

7.7.1 New features in protocol 2

- New onlinestatus SKYPEME.
- If call is set on hold, API notifies clients with "CALL xx STATUS LOCALHOLD / REMOTEHOLD" respectively. Protocol 1 simply returned ONHOLD.
- New call status CANCELLED

7.7.2 New features in protocol 3

- Multiperson chat commands
- Compatibility layer for previous IM

7.7.3 New features in protocol 4

- Language and country values are prefixed by ISO code

Protocol 1, 2 compatibility

If requested protocol is smaller than 3, then all incoming commands are converted as follows:

- SEARCH MESSAGES -> SEARCH CHATMESSAGES
- SEARCH MISSEDMESSAGES -> SEARCH MISSEDCHATMESSAGES
- GET MESSAGE -> GET CHATMESSAGE
- SET MESSAGE -> SET CHATMESSAGE

In addition are converted GET MESSAGE properties:

- PARTNER_HANDLE -> FROM_HANDLE
- PARTNER_DISPNAME -> FROM_DISPNAME

All API sent notifications' (including GET/SET MESSAGE) replies are converted:

- CHATMESSAGE * FROM_HANDLE x -> MESSAGE * PARTNER_HANDLE x
- CHATMESSAGE * FROM_DISPNAME x -> MESSAGE * FROM_DISPNAME x
- CHATMESSAGE * PROP x -> MESSAGE * PROP x

In case of protocol being smaller than 3, SEARCH MESSAGES and SEARCH MISSEDMESSAGES commands return string MESSAGES 1, 2, 3.

7.8 From Skype to Device

7.8.1 Status commands

All these commands are broadcasted by Skype after initial connection or if the parameter changes. They can be queried at any time with GET command.

User status

Syntax: *USERSTATUS status*
status - value for user status. Available values:

- **UNKNOWN**.
- **ONLINE** - current user is online.
- **OFFLINE** - current user is offline.
- **SKYPEME** - current user is in "Skype Me" mode (Protocol 2).
- **AWAY** - current user is away.
- **NA** - current user is not available.
- **DND** - current user is in "Do not disturb" mode.
- **INVISIBLE** - current user is invisible to others.
- **LOGGEDOUT** - current user is logged out. Clients are detached.

Example: *USERSTATUS INVISIBLE*

Connection status

Syntax: *CONNSTATUS status*
status - value for connection status. Available values:

- **OFFLINE**
- **CONNECTING**
- **PAUSING**
- **ONLINE**
- **LOGGEDOUT** - current user is logged out.

Example: *CONNSTATUS ONLINE*

Current user handle

Syntax: *CURRENTUSERHANDLE userhandle*
Example: *CURRENTUSERHANDLE banana*

7.8.2 Search results

These are responses to the SEARCH command.

Number of search results will be limited to N (N can be set) in the future, not limited at the moment. Possibly support for paging or custom range queries will be added later.

User search

List of found users.

Syntax: *USERS user1 [,user2] [,user3]*
Example: *USERS abc, -abc-, abc10*

Friend search

List of found friends.

Syntax: *USERS user1 [,user2] [,user3]*
Example: *USERS tim, joe, mike*

Call search
List of found call ID-s.

Syntax: *CALLS id1 [,id2] [,id3]*
Example: *CALLS 15, 16, 39*

Missed call search
List of found missed call ID-s.

Syntax: *CALLS id1 [,id2] [,id3]*
Example: *CALLS 15, 16, 39*

Message search
List of found message ID-s.

Syntax: *MESSAGES id1 [,id2] [,id3]*
Example: *MESSAGES 15, 16, 39*

Missed message search
List of found missed message ID-s.

Syntax: *MESSAGES id1 [,id2] [,id3]*
Example: *MESSAGES 15, 16, 39*

Chat search
List of found chat ID-s.

Syntax: *CHATS id1 [,id2] [,id3]*
Example: *CHATS #test_2/$testtest20;54389d65f7d6f2c4, #test_b/$testtest20;867c47704bcc71fb*

Active chat search
List of found chat ID-s that are open in UI.

Syntax: *CHATS id1 [,id2] [,id3]*
Example: *CHATS #test_2/$testtest20;54389d65f7d6f2c4, #test_b/$testtest20;867c47704bcc71fb*

Missed chat search
List of found chat ID-s that have unread messages in them.

Syntax: *CHATS id1 [,id2] [,id3]*
Example: *CHATS #test_2/$testtest20;54389d65f7d6f2c4, #test_b/$testtest20;867c47704bcc71fb*

Recent chat search
List of found recent chat ID-s.

Syntax: *CHATS id1 [,id2] [,id3]*
Example: *CHATS #test_2/$testtest20;54389d65f7d6f2c4, #test_b/$testtest20;867c47704bcc71fb*

Bookmarked chat search
List of found bookmarked chat ID-s.

Syntax: *CHATS id1 [,id2] [,id3]*

Example: *CHATS #test_2/$testtest20;54389d65f7d6f2c4, #test_b/$testtest20;867c47704bcc71fb*

7.8.3 Notifications

Notifications are sent by Skype either if the corresponding object changes, or if the value of the property is asked with GET command. Also, if the prop value is changed by SET command, the change is confirmed with a notification. Notifications are sent about relevant objects - i.e. users in the buddylist, active calls, active IMs.

The "PROP" is a property of corresponding object.

USER object
Notifies about user object properties.

Syntax: *USER USERNAME PROP VALUE*
USERNAME - username.
PROP - property name. Available properties are:

- **HANDLE** - username. Example: *USER pamela HANDLE pamela*.
- **FULLNAME** - user's full name. Example: *USER pamela FULLNAME Jane Doe*.
- **BIRTHDAY** - user's birth date YYYYMMDD. Example: *USER bitman BIRTHDAY 19780329*.
- **SEX** - example: *USER pamela SEX UNKNOWN*. Values:
  - **UNKNOWN** - user has not specified sex in personal profile.
  - **MALE**
  - **FEMALE**
- **LANGUAGE** - language's name. Example: *USER mike LANGUAGE English*. In protocol 4 with ISO prefix, example: *USER mike LANGUAGE en English*.
- **COUNTRY** - country's name. Example: *USER mike COUNTRY Estonia*. In protocol 4 with ISO prefix, example: *USER mike COUNTRY ee Estonia*.
- **PROVINCE** - example: *USER mike PROVINCE Harjumaa*.
- **CITY** - example: *USER mike CITY Tallinn*.
- **PHONE_HOME** - example: *USER mike PHONE_HOME 3721111111*.
- **PHONE_OFFICE** - example: *USER mike PHONE_OFFICE 3721111111*.
- **PHONE_MOBILE** - example: *USER mike PHONE_MOBILE 3721111111*.
- **HOMEPAGE** - example: *USER mike HOMEPAGE http://www.joltid.com*.
- **ABOUT** - example: *USER mike ABOUT I am a nice person*.
- **HASCALLEQUIPMENT** - returns always **TRUE**. Example: *USER pamela*

HASCALLEQUIPMENT TRUE.

- **BUDDYSTATUS** - example: *USER pamela BUDDYSTATUS 2*. Possible BUDDYSTATUS values:
  - **0** - never been in contact list.
  - **1** - deleted from contact list.
  - **2** - pending authorisation.
  - **3** - added to contact list.
- **ISAUTHORIZED** - is user authorized by current user. Example: *USER pamela ISAUTHORIZED TRUE*. Values:
  - **TRUE**
  - **FALSE**
- **ISBLOCKED** - is user blocked by current user. Example: *USER spammer ISBLOCKED TRUE*. Values:
  - **TRUE**
  - **FALSE**
- **DISPLAYNAME** - example: *USER pamela DISPLAYNAME pam*.
- **ONLINESTATUS** user online status. Example: *USER mike ONLINESTATUS ONLINE*. Values:
  - **UNKNOWN** - unknown user.
  - **OFFLINE** - user is offline (not connected). Will also be returned if current user is not authorized by other user to see his/her online status.
  - **ONLINE** - user is online.
  - **AWAY** - user is away (has been inactive for certain period).
  - **NA** - user is not available.
  - **DND** - user is in "Do not disturb" mode.
  - **SKYPEOUT** - user is in the SkypeOut contact list.
  - **SKYPEME** (Protocol 2)
- **LASTONLINETIMESTAMP** - UNIX timestamp, available only for offline user. Example *USER mike LASTONLINETIMESTAMP 1078959579*.

CALL object

Notifies about call object properties.

Syntax: *CALL ID PROP VALUE*

ID - call ID.

PROP - property name. Available properties are:

- **TIMESTAMP** - time, when call was placed (UNIX timestamp). Example: *CALL 17 TIMESTAMP 1078958218*
- **PARTNER_HANDLE** - example: *CALL 17 PARTNER_HANDLE mike*

- **PARTNER_DISPNAME** - example: *CALL 17 PARTNER_DISPNAME Mike Mann*
- **CONF_ID** - if CONF_ID>0 then it is conference call. Example: *CALL 17 CONF_ID 0*
- **TYPE** - call type. Example: *CALL 17 TYPE OUTGOING_PSTN*. Possible TYPE values:
    - **INCOMING_PSTN** - incoming call from PSTN.
    - **OUTGOING_PSTN** - outgoing call to PSTN.
    - **INCOMING_P2P** - incoming call from P2P.
    - **OUTGOING_P2P** - outgoing call to P2P.
- **STATUS** - call status. Example: *CALL 17 STATUS FAILED*. Possible STATUS values:
    - **UNPLACED** - call was never placed.
    - **ROUTING** - call is currently being routed.
    - **EARLYMEDIA** - with the pstn there is possibility that before the call is actually established, the early media is being played. For example it can be a calling tone or it can be some waiting message (all operators are busy, hold on for a sec) etc.
    - **FAILED** - call failed. Try to get FAILUREREASON for more information.
    - **RINGING** - currently ringing.
    - **INPROGRESS** - call is in progress.
    - **ONHOLD** - call is placed on hold.
    - **FINISHED** - call is finished.
    - **MISSED** - call was missed.
    - **REFUSED** - call was refused.
    - **BUSY** - destination was busy i.e. pressed hang up button.
    - **CANCELLED** (Protocol 2)
- **FAILUREREASON** - example: *CALL 17 FAILUREREASON 1* (numeric).
- **SUBJECT** - not used.
- **PSTN_NUMBER** - example: *CALL 17 PSTN_NUMBER 372123123*.
- **DURATION** - example: *CALL 17 DURATION 0*.
- **PSTN_STATUS** - error string from gateway, in case of PSTN-call. Example: *CALL 26 PSTN_STATUS 6500 PSTN connection creation timeout*.
- **CONF_PARTICIPANTS_COUNT** - number of non-hosts in case of conference call. Possible values:
    - **0** - call is not a conference. For host

CONF_PARTICIPANTS_COUNT is always 0.

- **1** - call is former conference.
- **2, 3, 4** - call is conference.

- **CONF_PARTICIPANT n** - conference's n-th participant's handle, call type and status and participant's displayname (for non-host only). Example: *CALL 59 CONF_PARTICIPANT 1 echo123 INCOMING_P2P INPROGRESS Echo Test Service*.

MESSAGE object

Notifies about message object properties.

Syntax: *MESSAGE ID PROP VALUE*

ID - message ID.

PROP - property name. Available properties are:

- **TIMESTAMP** - time, when message was sent (UNIX timestamp). Example: *MESSAGE 21 TIMESTAMP 1078958218*
- **PARTNER_HANDLE** - example: *MESSAGE 21 PARTNER_HANDLE mike*
- **PARTNER_DISPNAME** - example: *MESSAGE 21 PARTNER_DISPNAME Mike Mann*
- **CONF_ID** - not used.
- **TYPE** - message type. Example: *MESSAGE 21 TYPE TEXT*. Possible TYPE values:
  - **AUTHREQUEST** - authorization request.
  - **TEXT** - IM or topic set.
  - **CONTACTS** - contacts data.
  - **UNKNOWN** - other.
- **STATUS** - message status. Example: *MESSAGE 21 STATUS QUEUED*. Possible STATUS values:
  - **SENDING** - message is being sent.
  - **SENT** - message was sent.
  - **FAILED** - message sending failed. Try to get FAILUREREASON for more information.
  - **RECEIVED** - message has been received.
  - **READ** - message has been read.
  - **IGNORED** - message was ignored.
  - **QUEUED** - message is queued.
- **FAILUREREASON** - example: *MESSAGE 21 FAILUREREASON 1* (numeric).
- **BODY** - message body. Example: *MESSAGE 21 BODY Hi, what's up?*

CHATMESSAGE object

Notifies about chatmessage object properties.

Syntax: *CHATMESSAGE ID PROP VALUE*

ID - chatmessage ID.

PROP - property name. Available properties are:

- **TIMESTAMP** - time, when message was sent (UNIX timestamp). Example: *MESSAGE 21 TIMESTAMP 1078958218*
- **PARTNER_HANDLE** - example: *CHATMESSAGE 21 PARTNER_HANDLE mike*
- **PARTNER_DISPNAME** - example: *CHATMESSAGE 21 PARTNER_DISPNAME Mike Mann*
- **TYPE** - message type. Example: *MESSAGE 21 TYPE TEXT*. Possible TYPE values:
  - **SETTOPIC** - chat's topic change.
  - **SAID** - IM.
  - **ADDEDMEMBERS** - invited someone to chat.
  - **SAWMEMBERS** - chat participant has seen the other members.
  - **CREATEDCHATWITH** - chat to multiple people is created.
  - **LEFT** - someone left chat; also notification if somebody cannot be added to chat.
  - **UNKNOWN** - other.
- **STATUS** - message status. Example: *MESSAGE 21 STATUS QUEUED*. Possible STATUS values:
  - **SENDING** - message is being sent.
  - **SENT** - message was sent.
  - **RECEIVED** - message has been received.
  - **READ** - message has been read.
- **LEAVEREASON** - used with LEFT-type message. Example: *CHATMESSAGE 21 LEAVEREASON UNSUBSCRIBE*. Possible LEAVEREASON values:
  - **USER_NOT_FOUND** - user was not found.
  - **USER_INCAPABLE** - user has older Skype version and cannot join multichat.
  - **ADDER_MUST_BE_FRIEND** - recipient accepts messages from contacts only and sender is not in his/her Contact list.
  - **ADDED_MUST_BE_AUTHORIZED** - recipient accepts messages from authorized users only and sender is not authorized.
  - **UNSUBSCRIBE** - participant left chat.
- **BODY** - message body. Example: *CHATMESSAGE 21 BODY Hi, what's*

up?

- **CHATNAME** - chat that includes the message, example: #test_3/$b17eb511457e9d20
- **USERS** - people added to chat.

CHAT object (protocol 3)
Notifies about chat object properties.

Syntax: *CHAT ID PROP VALUE*
ID - chat ID.
PROP - property name. Available properties are:

- **NAME** - chat ID. Example: CHAT #test_l/$6a072ce5537c4044 NAME #test_l/$6a072ce5537c4044
- **TIMESTAMP** - time, when chat was created. Example: CHAT #test_l/$6a072ce5537c4044 TIMESTAMP 1078958218.
- **ADDER** - user who added current user to chat. Example: CHAT 1078958218 ADDER k6rberebane.
- **STATUS** - chat status. Example: CHAT #test_l/$6a072ce5537c4044 STATUS MULTI_SUBSCRIBED. Possible STATUS values:
  - **LEGACY_DIALOG** - old style IM.
  - **DIALOG** - 1:1 chat.
  - **MULTI_SUBSCRIBED** - participant in chat.
  - **UNSUBSCRIBED** - left from chat.
- **POSTERS** - members who have posted messages. Example: CHAT #test_l/$6a072ce5537c4044 POSTERS k6rberebane test_3
- **MEMBERS** - all users who have been there. Example: CHAT #test_l/$6a072ce5537c4044 MEMBERS k6rberebane test test_2 test_3
- **TOPIC** - chat topic. Example: CHAT #test_l/$6a072ce5537c4044 TOPIC API testimine
- **CHATMESSAGES** - all messages ID-s in this chat. Example: CHAT #test_l/$6a072ce5537c4044 CHATMESSAGES 34, 35, 36, 38, 39
- **ACTIVEMEMBERS** - members who have stayed in chat. Example: CHAT #test_l/$6a072ce5537c4044 ACTIVEMEMBERS k6rberebane test_2 test_3
- **FRIENDLYNAME** - name shown in chat window title. Example: CHAT #test_l/$6a072ce5537c4044 FRIENDLYNAME Test Test XX | tere ise ka

CALL HISTORY
Notifies about call history being changed and that it needs to be reloaded. Happens, when all of the call history or a selection of it has been deleted.

Syntax: *CALLHISTORYCHANGED*

IM HISTORY
Notifies about instant message history being changed and that it needs to be

reloaded. Right now it only happens, when all of the IM history is deleted.

Syntax: *IMHISTORYCHANGED*

BUDDYSTATUS
Notifies if some user is added to or deleted from contacts or has authorized current user.

Syntax: *USER username BUDDYSTATUS number*
User has been added to contacts, pending authorisation.
USER pamela BUDDYSTATUS 2
Example: User has authorized current user.
USER pamela BUDDYSTATUS 3
User has been deleted from contacts.
USER pamela BUDDYSTATUS 1

7.8.4 Other

Input-output devices can be set for Skype. Setting device with empty name selects Windows default device. Successful setting is confirmed with AUDIO_* [device].

Current active devices can be queried with *GET AUDIO_IN|AUDIO_OUT*. AUDIO_* will return an empty answer, when the default device is selected.

Audio input device
Syntax: *AUDIO_IN[device name]* **(deprecated)**
SET AUDIO_IN[device name]
Example: *AUDIO_IN SB Audigy 2 ZS Audio [DC00]* **(deprecated)**
SET AUDIO_IN SB Audigy 2 ZS Audio [DC00]

Audio output device
Syntax: *AUDIO_OUT[device name]* **(deprecated)**
SET AUDIO_OUT[device name]
Example: *AUDIO_OUT SB Audigy 2 ZS Audio [DC00]* **(deprecated)**
SET AUDIO_OUT SB Audigy 2 ZS Audio [DC00]

7.9 From Device to Skype

7.9.1 Initiating searches

SEARCH WHAT [target] requests a specific type of information about the target. If no target is specified, then all results are returned.

WHAT specifies the information type and may be one of the following: **USERS, FRIENDS, CALLS, MISSEDCALLS, ACTIVECALLS, MESSAGES, MISSEDMESSAGES, (protocol 3: CHATS, ACTIVECHATS, MISSEDCHATS, RECENTCHATS, BOOKMARKEDCHATS, CHATMESSAGES, MISSEDCHATMESSAGES)**.

Note: Currently search commands are synchronous. This may change in the future.

Note: Next search expires last search. This means that if one search is still running, and new search is submitted, the first search will abort.

Search friends

Syntax: *SEARCH FRIENDS*

Returns: Returns a list of usernames, if match is found.

Example: *SEARCH FRIENDS*
Returns all friends of the current user. Example result:
USERS tim, joe, mike

Errors: *ERROR 67 target not allowed with SEARCH FRIENDS*
Target was specified with SEARCH FRIENDS command (e.g. "SEARCH FRIENDS mike")

Search users

Syntax: *SEARCH USERS TARGET*

TARGET - username. If the search string contains "@", then search is performed by e-mail address (note that e-mail address has to match 100%). Otherwise, if the search string is a valid Skype username (username must have 6-22 characters, contains only the following symbols: **a-Z0-9-_,.** and it must start with a letter), the search is performed on the full name and username fields. In all other cases the search is made on full name field only.

Returns: Returns the matching usernames.

Example: *SEARCH USERS abc*
Returns all usernames that have "abc" in them. Example result:
USERS abc, -abc-, abc10

Errors: *ERROR 4 empty target not allowed*
Target username is not specified

Search calls

Syntax: *SEARCH CALLS TARGET*

TARGET - username. Target is optional. If target is specified then call history between current user and target user is searched.

Returns: Returns a list of call ID-s. If target is specified, then returns ID-s of all calls that have been made between current and target user.

Example: *SEARCH CALLS abc*
Example result:
CALLS 15, 16, 39

Errors: *ERROR 5 SEARCH CALLS: invalid target*
Not permitted characters were used in the target username (e.g. "SEARCH CALLS !a"). Username must have 6-22 characters and can contain only the following symbols: **a-Z0-9-_,.**.

Search active calls

Syntax: *SEARCH ACTIVECALLS*

Lists all calls visible on calltabs, including members of conference calls if hosting a conference.

Returns: Returns a list of active call ID-s.

Example: *SEARCH ACTIVECALLS*
Example result:
CALLS 25, 56

Errors: *ERROR 3 SEARCH: invalid WHAT*
ACTIVECALLS was misspelled.

Search missed calls

Syntax: *SEARCH MISSEDCALLS*

Returns: Returns a list of missed call ID-s, calls in MISSED status.

Example: *SEARCH MISSEDCALLS*
Example result:
CALLS 25, 56

Errors: *ERROR 29 target not allowed with MISSEDCALLS*
No target is allowed with SEARCH MISSEDCALLS.

Search messages

Syntax: *SEARCH MESSAGES [TARGET]*

TARGET - username. Target is optional. If target is specified then messages history between current user and target user is searched.

Returns: Returns a list of message ID-s. If target is specified, then returns ID-s of all messages that have been sent between current and target user.

Example: *SEARCH MESSAGES abc*
Example result:
MESSAGES 123, 124

Errors: *ERROR 29 SEARCH MESSAGES: invalid target*
Not permitted character was used in the target username (e.g. "SEARCH MESSAGES !a"). Username must have 6-22 characters and can contain only the following symbols: **a-Z0-9-_,.**.

Search missed messages

Syntax: *SEARCH MISSEDMESSAGES*

Returns: Returns a list of message ID-s.

Example: *SEARCH MISSEDMESSAGES*
Example result:
MESSAGES 123, 124

Errors: *ERROR 6 target not allowed with MISSEDMESSAGES*
No target is allowed with SEARCH MISSEDMESSAGES.

Search chats (PROTOCOL 3)

Syntax: *SEARCH CHATS*

Returns: Returns a list of chat ID-s.

Example: *SEARCH CHATS*
Example result:
CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391

Errors: *ERROR 107 target not allowed with CHATS*
No target is allowed with SEARCH CHATS.

Search active chats (PROTOCOL 3)

Syntax: *SEARCH ACTIVECHATS*

Returns: Returns a list of chat ID-s that are open in UI.

Example: *SEARCH ACTIVECHATS*
Example result:
CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391

Errors: ?

Search missed chats (PROTOCOL 3)

Syntax: *SEARCH MISSEDCHATS*

Returns: Returns a list of chat ID-s that include unread messages.

Example: *SEARCH MISSEDCHATS*
Example result:
CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391

Errors: ?

Search recent chats (PROTOCOL 3)

Syntax: *SEARCH RECENTCHATS*

Returns: Returns a list of recent chat ID-s.

Example: *SEARCH RECENTCHATS*
Example result:
CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391

Errors: ?

Search bookmarked chats (PROTOCOL 3)

Syntax: *SEARCH BOOKMARKEDCHATS*

Returns: Returns a list of bookmarked chat ID-s.

Example: *SEARCH BOOKMARKEDCHATS*
Example result:
CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391

Errors: ?

Search chat messages (PROTOCOL 3)

Syntax: *SEARCH CHATMESSAGES [TARGET]*
TARGET - username. Target is optional. Actually users never use it.

Returns: Returns a list of chat message ID-s.

Example: *SEARCH CHATMESSAGES abc*
Example result:
CHATMESSAGES 60, 59

Errors: *ERROR 29 SEARCH CHATMESSAGES: invalid target*
Not permitted character was used in the target username (e.g. "SEARCH MESSAGES !a"). Username must have 6-22 characters and can contain only the following symbols: **a-Z0-9-_,.**.

Search missed chat messages (PROTOCOL 3)

Syntax: *SEARCH MISSEDCHATMESSAGES*

Returns: Returns a list of missed chat message ID-s.

Example: *SEARCH MISSEDCHATMESSAGES*
Example result:
CHATMESSAGES 61, 62

Errors: *ERROR 29 target not allowed with MISSEDMESSAGES*
No target is allowed with SEARCH MISSEDCHATMESSAGES.

7.9.2 Getting parameter value

GET command is a general request command. One can request one property at a time for any known object (**USER, CALL, MESSAGE, (PROTOCOL 3: CHAT, CHATMESSAGE)**) or a general variable (**USERSTATUS, CONNSTATUS, AUDIO_IN, AUDIO_OUT, CURRENTUSERHANDLE, MUTE**). Skype responds with appropriate notification command.

USER object information

Syntax: *GET USER USERNAME PROP*
USERNAME - username. Username must have 6-22 characters and can only contain the following symbols: **a-Z0-9-_,.**.
PROP - property name. Available properties are:
HANDLE, FULLNAME, BIRTHDAY, SEX, LANGUAGE, COUNTRY, PROVINCE, CITY, PHONE_HOME, PHONE_OFFICE, PHONE_MOBILE, HOMEPAGE, ABOUT, HASCALLEQUIPMENT, BUDDYSTATUS, ISAUTHORIZED, ISBLOCKED, DISPLAYNAME, ONLINESTATUS, LASTONLINETIMESTAMP

Returns: Returns property value for specified user, if match is found.

Example: *GET USER pamela FULLNAME*
Example result:
USER pamela FULLNAME Jane Doe

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET USE").
ERROR 10 invalid prop
ID and/or property missing or misspelled (e.g. "GET USER pamela FULLNAM").
ERROR 8 invalid handle
USERNAME missing or includes a not permitted character (e.g. "GET USER ! HANDLE").

Note *GET USER USERNAME ONLINESTATUS* will return "OFFLINE" unless current user is authorized by other user to see his/her online status.

CALL object information

Syntax: *GET CALL ID PROP*

ID - call ID

PROP - property name. Available properties are:

TIMESTAMP (UNIX timestamp), PARTNER_HANDLE, PARTNER_DISPNAME, CONF_ID, TYPE, STATUS, FAILUREREASON (numeric), SUBJECT (not used), PSTN_NUMBER, DURATION, PSTN_STATUS, CONF_PARTICIPANT n, CONF_PARTICIPANTS_COUNT

Returns: Returns property value for specified call, if call is found.

Example: *GET CALL 1594 TYPE*
Example result:
CALL 1594 TYPE OUTGOING_P2P

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET CAL").
ERROR 11 invalid call id
ID includes other than numeric characters (e.g. "GET CALL 15!").
ERROR 12 unknown call id
Call with specified ID does not exist in current user's call history.
ERROR 13 invalid prop
Property name missing or misspelled (e.g. "GET CALL 15 TYP").
ERROR 71 Invalid conference participant NO
Conference participant's number is not number or too big (e.g. "GET CALL 15 CONF_PARTICIPANT kala", "GET CALL 15 CONF_PARTICIPANT 5")

Note Skype notifies API, if unseen call gets seen i.e. SEEN=FALSE changes to SEEN=TRUE e.g. user clicks on the missed call in Skype.

MESSAGE object information

Syntax: *GET MESSAGE ID PROP*

ID - message ID

PROP - property name. Available properties are:

TIMESTAMP (UNIX timestamp), PARTNER_HANDLE, PARTNER_DISPNAME, CONF_ID (not used), TYPE, STATUS, FAILUREREASON (numeric), BODY

Returns: Returns property value for specified message, if message is found.

Example: *GET MESSAGE 159 TYPE*
Example result:
MESSAGE 159 TYPE TEXT

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET MESAGE").
ERROR 14 invalid message id
ID includes other than numeric characters (e.g. "GET MESSAGE 1a").
ERROR 15 unknown message id
Message with specified ID does not exist in current user's message history.
ERROR 16 invalid prop
Property name missing or misspelled (e.g. "GET MESSAGE 21 TYP").

User status

Syntax: *GET USERSTATUS*

Returns: Returns status for current user. Possible values: **UNKNOWN, OFFLINE, ONLINE, SKYPEME (Protocol 2), AWAY, NA, DND, INVISIBLE.**

Example: *GET USERSTATUS*
Example result:
USERSTATUS ONLINE

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET USESTATUS").

Connection status

Syntax: *GET CONNSTATUS*

Returns: Returns current connection status. Possible values: **OFFLINE, CONNECTING, PAUSING, ONLINE.**

Example: *GET CONNSTATUS*
Example result:
CONNSTATUS ONLINE

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET CONSTATUS").

Audio input device

Syntax: *GET AUDIO_IN*

Returns: Returns current audio input device for Skype.

Example: *GET AUDIO_IN*
Example result:
AUDIO_IN SB Audigy 2 ZS Audio [DC00]

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET AUDO_IN").

Audio output device

Syntax: *GET AUDIO_OUT*

Returns: Returns current audio output device for Skype.

Example: *GET AUDIO_OUT*
Example result:
AUDIO_OUT SB Audigy 2 ZS Audio [DC00]

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET AUDIO_OT").

Current user handle

Syntax: *GET CURRENTUSERHANDLE*

Returns: Returns handle for current user.

Example: *GET CURRENTUSERHANDLE*
Example result:
CURRENTUSERHANDLE banana

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET CURENTUSERHANDLE").

Mute status

Syntax: *GET MUTE*

Returns: Returns mute status: **MUTE ON** or **MUTE OFF**. If there is no calls in status INPROGRESS, MUTE is always **OFF**.

Example: *GET MUTE*
Example result:
MUTE OFF

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET MUT").

User privileges

Syntax: *GET PRIVILEGE [PRIVILEGE]*

PRIVILEGE - privilege name. Available privileges are:
SKYPEOUT

Returns: Returns **TRUE**, if user has privilege, and **FALSE** otherwise.

Example: *GET PRIVILEGE SKYPEOUT*
Example result:
PRIVILEGE SKYPEOUT TRUE

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET PIVILEGE").
ERROR 40 unknown privilege
Privilege is either misspelled or does not exist (e.g. "GET PRIVILEGE SKYPEOUT").

User profile

Syntax: *GET PROFILE [PROP]*

PROP - profile property. Available properties are:
PSTN_BALANCE - SkypeOut balance in EUR cents.
PSTN_BALANCE_CURRENCY - SkypeOut currency. Only possible currency is EUR.

Returns: PSTN_BALANCE_CURRENCY returns **PROFILE PSTN_BALANCE_CURRENCY EUR**, if user has SkypeOut privilege, and **PROFILE PSTN_BALANCE_CURRENCY** otherwise.

Example: *GET PROFILE PSTN_BALANCE*
Example result:
PROFILE PSTN_BALANCE 109

Example: *GET PROFILE PSTN_BALANCE_CURRENCY*
Example result:
PROFILE PSTN_BALANCE_CURRENCY EUR

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET PROFIIL").
ERROR 10 Invalid PROP
Property is either misspelled or does not exist (e.g. "GET PROFILE PSTN_BALANSS").

Skype version

Syntax: *GET SKYPEVERSION*

Returns: Returns Skype version.

Example: *GET SKYPEVERSION*
Example result:
SKYPEVERSION 1.0.0.34

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET SKYPVERSION").

CHAT object information (PROTOCOL 3)

Syntax: *GET CHAT ID PROP*

ID - chat ID.
PROP - property name. Available properties are:
NAME, TIMESTAMP, ADDER, STATUS, POSTERS, MEMBERS, TOPIC, CHATMESSAGES, ACTIVEMEMBERS, FRIENDLYNAME

Returns: Returns property value for specified chat, if match is found. STATUS values:
LEGACY_DIALOG, DIALOG, MULTI_SUBSCRIBED, UNSUBSCRIBED

Example: *GET CHAT #bitman/$jessy;eb06e65635359671 NAME*
Example result:
CHAT #bitman/$jessy;eb06e65635359671 NAME #bitman/$jessy;eb06e65635359671

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET CHTA ").
ERROR 105 invalid chat name
Error in the CHATNAME parameter.
ERROR 106 invalid PROP
Property name missing or misspelled.

CHATMESSAGE object information (PROTOCOL 3)

Syntax: *GET CHATMESSAGE ID PROP*

ID - chat message ID.
PROP - property name. Available properties are:
CHATNAME, TIMESTAMP, FROM_HANDLE, FROM_DISPNAME, TYPE, USERS, LEAVEREASON, BODY, STATUS

Returns: Returns property value for specified chat message, if match is found.
TYPE values:
SETTOPIC, SAID, ADDEDMEMBERS, SAWMEMBERS, CREATEDCHATWITH, LEFT, UNKNOWN
LEAVEREASON values:
USER_NOT_FOUND, USER_INCAPABLE, ADDER_MUST_BE_FRIEND, ADDED_MUST_BE_AUTHORIZED, UNSUBSCRIBE
STATUS values:
SENDING, SENT, RECEIVED, READ

Example: *GET CHATMESSAGE 60 CHATNAME*
Example result:
CHATMESSAGE 60 CHATNAME
#bitman/$jessy;eb06e65631239671

Errors: *ERROR 7 GET: invalid WHAT*
Object name missing or misspelled (e.g. "GET CHTAMESSAGE 60 CHATNAME").
ERROR 14 invalid message id
Chat message ID contain not permitted symbols (only numeric are permitted) (e.g. "GET CHATMESSAGE a")
ERROR 15 unknown message id
Unknown chat message ID
ERROR 16 invalid PROP
Property name missing or misspelled (e.g. "GET CHATMESSAGE 60")

7.9.3 Setting parameter value

SET command is a general update command. Syntactically it can be applied to objects (**CALL, USER, MESSAGE**) and general variables (**USERSTATUS**). One can update one property at a time for any known object. Skype responds with appropriate notification command as a confirmation.

Note: Most properties are read-only.

CALL object information

Syntax: *SET CALL ID PROP VALUE*
ID - call ID (numeric)
PROP - property name. Writable properties for call:

- **STATUS** - for call control. Available values:
  - **ONHOLD** - hold call
  - **INPROGRESS** - answer or resume call
  - **FINISHED** - hang up call
- **SEEN** - sets call seen, meaning that this missed call is seen and will be removed from missed calls list.
- **DTMF** - sends VALUE as DTMF. Permitted symbols in VALUE are: {0..9,#,*}.
- **JOIN_CONFERENCE** - joins call with another call into conference.

VALUE is another call's ID.

Returns: Returns (new) value of the property or an error message.

Example: *SET CALL 15 SEEN*
Example result:
CALL 15 SEEN TRUE

Errors: *ERROR 18 SET: invalid WHAT*
Object name missing or misspelled (e.g. "SET CAL").
ERROR 19 invalid call id
ID includes other than numeric characters (e.g. "SET CALL A").
ERROR 20 unknown call id
Call with specified ID does not exist in current user's call history nor is active.
ERROR 21 unknown prop
Value incorrect or misspelled (e.g. "SET CALL 15 STATUS ONHOL").
ERROR 22 cannot hold
Given call is not in progress and therefore can not be hold.
ERROR 23 cannot resume/answer
Given call is not in progress and therefore can not be resumed/answered.
ERROR 24 cannot hangup
Given call is not in progress and therefore can not be hung up.
ERROR 25 invalid WHAT
Property name missing or misspelled (e.g. "SET CALL 15 STATU ONHOLD").
ERROR 72 Cannot create conference
Creating conference, e.g. "SET CALL 65 JOIN_CONFERENCE 66" fails on some reason.

MESSAGE object information

Syntax: *SET MESSAGE ID PROP*

ID - message ID (numeric)
PROP - property name. Writable properties for message:

- **SEEN** - meaning that this missed IM is seen and will be removed from missed messages list. UI will set this automatically if auto-popup is enabled for corresponding group of users.

Returns: Returns (new) value of the property or an error message. If value can not be changed then the current value of property is returned.

Example: *SET MESSAGE 1578 SEEN*
Example result:
MESSAGE 1578 STATUS READ

Errors: *ERROR 18 SET: invalid WHAT*
Object name missing or misspelled (e.g. "SET CAL").
ERROR 30 invalid message id
ID includes other than numeric characters (e.g. "SET MESSAGE A").
ERROR 31 unknown message id
Message with specified ID does not exist in current user's message history.
ERROR 32 invalid WHAT
Property name missing or misspelled (e.g. "SET MESSAGE 21 SEN").

User status

Syntax: *SET USERSTATUS VALUE*

VALUE - value for user status. Available values:

- **ONLINE** - set current user status "online"
- **OFFLINE** - set current user status "offline"
- **SKYPEME** - set current user status "Skype Me" (Protocol 2)
- **AWAY** - set current user status "away"
- **NA** - set current user status "NA"
- **DND** - set current user status "DND"
- **INVISIBLE** - set current user status "INVISIBLE"

Returns: Returns (new) value of the property or an error message.

Example: *SET USERSTATUS INVISIBLE*
Example result:
USERSTATUS INVISIBLE

Errors: *ERROR 18 SET: invalid WHAT*
USERSTATUS command missing or misspelled (e.g. "SET CAL").
ERROR 28 SET USERSTATUS: unknown status
Unknown or misspelled value for user status (e.g. "SET USERSTATUS RICH").

Mute status

Syntax: *SET MUTE VALUE*

VALUE - Sets mute on or off. Works only if call is in status INPROGRESS. Available values:

- **ON** - set mute on
- **OFF** - set mute off

Returns: Returns (new) value of the property or an error message.

Example: *SET MUTE ON*
Example result:
MUTE ON

Errors: *ERROR 18 SET: invalid WHAT*
Mute command missing or misspelled (e.g. "SET MUT").
ERROR 33 invalid parameter
Unknown or misspelled value for mute (e.g. "SET MUTE O").

Chat message status

Syntax: *SET CHATMESSAGE ID SEEN*
ID - chat message ID.

Returns: Returns (new) value for chat message status.

Example: *SET CHATMESSAGE 61 SEEN*
Example result:
CHATMESSAGE 61 STATUS SEEN

Errors: *ERROR 18 SET: invalid WHAT*
CHATMESSAGE command missing or misspelled (e.g. "SET CHATMESAGE").
ERROR 31 unknown message id
Unknown chat message ID
ERROR 30 invalid message id
Chat message ID is misspelled i.e. contains not permitted symbols (numeric are permitted) (e.g. "SET CHATMESSAGE a SEEN")
ERROR 32 invalid WHAT
Invalid status given to chat message (e.g. "SET CHATMESSAGE 60 SEENA")

7.9.4 Making calls

Syntax: *CALL TARGET [, TARGET2, TARGET3...]*
TARGET - target to be called. In case of multiple targets conference is created. Available target types:

- **USERNAME** - username, e.g. "pamela"
- **PSTN** - phone number, e.g. "003725555555"
- **SPEED DIAL CODE** - 1 or 2 character speed-dial code

Returns: Returns call ID and status.

Example: *CALL pamela*
Example result:
CALL 49 STATUS ROUTING
CALL 49 STATUS RINGING
CALL 49 STATUS REFUSED

Errors: *ERROR 34 invalid user handle*
Target username/number missing (e.g. "CALL ").
ERROR 39 user blocked
Not really a call failure, but API misuse - trying to call to a blocked user.
ERROR 73 too many participants
Call is initiated to more than 4 people (e.g. "CALL test, test_2, test_3, test_4, test_5").

Skype window is focused when call is initiated through API. It is possible to make speed dial via API.

Call error codes

Code	Description	Possible reason
1	CALL 181 FAILUREREASON 1	Misc error.
2	CALL 181 FAILUREREASON 2	User/phone number does not exist. Check phone number prefix (e.g. correct "003725555555", "+3725555555"; incorrect "3725555555").
3	CALL 181 FAILUREREASON 3	User is offline.
4	CALL 181 FAILUREREASON 4	No proxy found.
5	CALL 181 FAILUREREASON 5	Session terminated.
6	CALL 181 FAILUREREASON 6	No common codec found.
7	CALL 181 FAILUREREASON 7	Sound I/O error.
8	CALL 181 FAILUREREASON 8	Problem with remote sound device.
9	CALL 181 FAILUREREASON 9	Call blocked by recipient.
10	CALL 181 FAILUREREASON 10	Recipient not a friend.
11	CALL 181 FAILUREREASON 11	Current user not authorized by recipient.
12	CALL 181 FAILUREREASON 12	Sound recording error.

7.9.5 Sending messages

Syntax: *MESSAGE USERNAME TEXT*
USERNAME - username, whom to send message, e.g. "pamela"
TEXT - message body, e.g. "Please call me"

Returns: Returns message ID and status.

Example: *MESSAGE pamela Please call me*
Example result:
MESSAGE 136 STATUS SENDING
MESSAGE 136 STATUS SENT

Errors: *ERROR 26 invalid user handle*
Target username missing or includes not permitted symbols (e.g. "MESSAGE ")
MESSAGE 138 STATUS RECEIVED
When message sending fails, a LEFT-type message is received. Message's LEAVEREASON shows why it failed.

7.9.6 Opening dialogs

Add a Contact

Syntax: *OPEN ADDAFRIEND*

Example: *OPEN ADDAFRIEND*
Add a Contact dialog will pop up.

Errors: *ERROR 69 invalid open what*
Open target is missing or misspelled (e.g. "OPEN ADDFRIEND").

Command is echoed back to API when successful.

Instant Message

Syntax: *OPEN IM USERNAME [MESSAGE]*

USERNAME - username, whom to send the message, e.g. "pamela"
MESSAGE - message body, e.g. "Please call me". It will be prefilled into the IM dialog window

Example: *OPEN IM jdenton Testing*
IM dialog will pop up, with text "Testing" already filled in as message text.

Errors: *ERROR 69 invalid open what*
Open target is missing or misspelled (e.g. "OPEN IN").
ERROR 70 SET: invalid handle
Username is missing or contains not permitted symbols

Command is echoed back to API when successful.

Chat

Syntax: *OPEN CHAT TARGET [, TARGET2, TARGET3 ...]*

TARGET - username to include to chat, e.g. "pamela". If only one TARGET is given, a dialog in opened

Returns: *CHAT id STATUS SUBSCRIBED* when successful.

Example: *OPEN CHAT jdenton, test_bi, pamela*
"CHAT #name/$843934 STATUS SUBSCRIBED", Chat window will pop up, with specified members in the right.

Errors: *ERROR 69 invalid open what*
??.
ERROR 70 SET: invalid handle
Username is missing or contains not permitted symbols

Command is echoed back to API when successful.

7.9.7 Focus Skype window

Syntax: *FOCUS*

Example: *FOCUS*
Brings Skype window on top.

Errors: ERROR 2 unknown command
FOCUS command is missing or misspelled (e.g. "FCOUS").

7.9.8 Test connection

Syntax: *PING*

Returns: *PONG* if Skype is present.

Example: *PING*
Query connection status.

Errors: ERROR 2 unknown command
PING command is missing or misspelled (e.g. "PNIG").

7.10 Error codes

- ERROR CODE[DESC]

The error response is sent by Skype each time Skype senses an error condition, which includes syntactically incorrect commands, internal inconsistencies etc. CODE is a number that uniquely identifies error condition and the DESC is optional brief description of the situation, given in English.

Currently the following error codes are defined:

Code	Description	Possible reasons
1	General syntax error	Command missing (e.g. " " sent as command)
2	Unknown command	Command spelled incorrect (e.g. "GRT" send instead of "GET")
3	Search: unknown WHAT	Search target is missing or misspelled
4	Empty target not allowed	
5	Search CALLS: invalid target	Not permitted character (e.g. "!", "#", "¤", "€", " " (space) etc.) was used in the target username (e.g. "SEARCH CALLS !a")
6	Target not allowed with MISSEDCALLS	e.g. "SEARCH MISSEDCALLS echo123"
7	GET: invalid WHAT	Object/property name missing or misspelled
8	Invalid user handle	USERNAME missing or includes a not permitted character (e.g. "GET USER ! HANDLE")
9	Unknown user	

10	Invalid PROP	Property name and/or ID missing or misspelled
11	Invalid call id	Call ID missing or misspelled (must be a numeric value)
12	Unknown call	Nonexistant call ID used
13	Invalid PROP	Returned to command GET CALL id PARTNER_DISPLAYNAME. Property name missing or misspelled
14	Invalid message id	GET - Message ID missing or misspelled (must be a numeric value)
15	Unknown message	Nonexistant message ID used in GET command
16	Invalid PROP	Returned to command GET MESSAGE id PARTNER_DISPLAYNAME. Property name missing or misspelled
17	Not in use	
18	SET: invalid WHAT	Property name missing or misspelled
19	Invalid call id	Call ID missing or misspelled (must be a numeric value)
20	Unknown call	Nonexistant call ID used
21	Unknown/disallowed call prop	SET CALL value incorrect or misspelled (e.g. "SET CALL 15 STATUS ONHOL")
22	Cannot hold this call at the moment	Trying to hold a call that is not in progress.
23	Cannot resume this call at the moment	Trying to resume/answer a call that is not in progress.
24	Cannot hangup inactive call	Trying to hang up a call that is not in progress.
25	Unknown WHAT	Property name missing or misspelled (e.g. "SET CALL 15 STATU ONHOLD")
26	Invalid user handle	Target username missing or includes not permitted symbols (e.g. "MESSAGE ")
27	Invalid version number	
28	Unknown userstatus	Unknown or misspelled value for user status (e.g. "SET USERSTATUS RICH")
29	Target not allowed with MISSEDMESSAGES	e.g. "SEARCH MISSEDMESSAGES echo123"
30	Invalid message id	SET - Message ID missing or misspelled (must be a numeric value)
31	Unknown message id	Nonexistant message ID used in SET command
32	Invalid WHAT	Prop missing or misspelled
33	Invalid parameter to SET MUTE	Unknown or misspelled value for mute (e.g. "SET MUTE O")

34	Invalid user handle to CALL	Target username/number missing (e.g. "CALL ")
35	Not connected	
36	Not online	
37	Not connected	
38	Not online	
39	User blocked	Destination user is blocked by caller. Also given, if trying to call to a blocked user
40	Unknown privilege	Privilege is either misspelled or does not exist (e.g. "GET PRIVILEGE SKYPEOUT").
41	Call not active	Trying to send DTMF, when call is not active.
42	Invalid DTMF code	Invalid DTMF code is sent. Valid symbols for DTMF codes are {0..9,#,*}
43	cannot send empty message	Empty message is tried to sent, e.g. "MESSAGE echo123".
66	Not connected	Skype is not connected i.e. user status is "LOGGEDOUT"
67	Target not allowed with SEARCH FRIENDS	SEARCH FRIENDS had a parameter
68	Access denied	
69	Invalid open what	OPEN command had missing or misspelled TARGET e.g. "OPEN IN"
70	Invalid handle	OPEN IM parameter USERNAME is missing or contains not permitted symbols
71	Invalid conference participant NO	Conference participant's number is either too large or invalid.
72	Cannot create conference	
73	too many participants	Conference is initiated to more than 4 people.
91	Internal error	Cannot call an emergency number
92	Internal error	The called number is not a valid PSTN number
93	Internal error	Invalid Skype Name
94	Internal error	Cannot call yourself
95	Internal error	Destination user is blocked by caller right after call initialization
96	Internal error	An outgoing call exists in ROUTING/RINGING/EARLYMEDIA state
97	Internal error	Internal error
98	Internal error	Internal error
99	Internal error	Internal error
100	Internal error	Internal error
101	Internal error	A call to the destination user is already ongoing

103	Cannot hold	Internal error
104	Cannot resume	Internal error
105	Invalid chat name	Chat name missing or misspelled
106	Invalid PROP	Property name missing or misspelled for CHAT or CHATMESSAGE
107	Target not allowed with CHATS	No parameters allowed to SEARCH CHATS
9901	Internal error	

7.11 Sample Session

-> Device to Skype
<- Skype to Device
; comment

NB! Real transmission goes utf-8 encoded.

; *Devices introduces himself & declares that it's ready*
-> NAME HandSet 1.0
-> PROTOCOL 1
; *-> READY (not implemented yet)*
; *Skype accepts/activates the device*
<- PROTOCOL 1
; *<- ENABLE YES (not implemented yet)*
; *Device asks for friends list*
-> SEARCH FRIENDS
; *Skype responds with the list*
<- USERS kaido, taavet, toivo
; *Device asks for fullname for all those in the friendslist*
-> GET USER kaido FULLNAME
-> GET USER taavet FULLNAME
-> GET USER toivo FULLNAME
; *Skype responds with data what was asked ...*
<- USER kaido FULLNAME Kaido KΣrner
<- USER taavet FULLNAME
<- USER toivo FULLNAME Toivo Annus
; *Device initiates a call to kaido*
-> CALL kaido
; *Skype indicates different call statuses*
<- CALL 122211 STATUS ROUTING
; *In the meantime, taavet went offline*
<- USER taavet ONLINESTATUS OFFLINE
<- CALL 122211 STATUS RINGING
; *Kaido picked up the call*
<- CALL 122211 STATUS INPROGRESS
; *Call is in progress ..*
; *Now device decides to hang up the call*
-> SET CALL 122211 STATUS FINISHED
; *Skype confirms ..*
<- CALL 122211 STATUS FINISHED

7.12Sample Session: incoming call

-> Device to Skype
<- Skype to Device
; comment

; *Skype notifies of incoming call*
<- CALL 1594 STATUS RINGING
; *Device queries call type*
-> GET CALL 1594 TYPE
<- CALL 1594 TYPE INCOMING_P2P
; *Device queries callers username and fullname after this*
-> GET CALL 1594 PARTNER_HANDLE
<- CALL 1594 PARTNER_HANDLE caller
-> GET USER caller FULLNAME
<- USER caller FULLNAME answer me
; *Device accepts (answers the call)*
-> SET CALL 1594 STATUS INPROGRESS
; *Skype confirms*
<- CALL 1594 STATUS INPROGRESS
; *Skype tells the duration of the call (this is done before the call is ended*
<- CALL 1594 DURATION 5
; *Skype tells that the other party ended the call*
<- CALL 1594 STATUS FINISHED

8 API TRANSPORT LAYERS

Both APIs are not bound to any specific transport-layer.

8.1API Transport on Windows messages

Currently the only API transport implementation is done on windows messages. In the current implementation the external party is active and is responsible for initiating the communication. The communication should be considered broken when either party could not send any of the messages.

To initiate communication, Client should broadcast windows message ('SkypeControlAPIDiscover') to all windows in the system, specifying its own window handle in wParam parameter. In response, Skype responds with message 'SkypeControlAPIAttach' to the handle specified, and indicates connection status with one of the following values in lParam:

- SKYPECONTROLAPI_ATTACH_SUCCESS = 0: Client is successfully attached and API window handle can be found in wParam parameter;
- SKYPECONTROLAPI_ATTACH_PENDING_AUTHORIZATION = 1: Skype has acknowledged connection request and is waiting for confirmation from the user. The client is not yet attached and should wait for SKYPECONTROLAPI_ATTACH_SUCCESS message;
- SKYPECONTROLAPI_ATTACH_REFUSED = 2: User has explicitly denied access to client;
- SKYPECONTROLAPI_ATTACH_NOT_AVAILABLE = 3: API is not

available at the moment. For example, this happens when no user is currently logged in. Client should wait for SKYPECONTROLAPI_ATTACH_API_AVAILABLE broadcast before making any further connection attempts.

When API becomes available, Skype broadcasts SkypeControlAPIAttach message with value SKYPECONTROLAPI_ATTACH_API_AVAILABLE = 0x8001 to all application windows in the system.

The actual data exchange uses standard WM_COPYDATA message. The data being transferred is the command (or response), given as an null-terminated utf-8 string. Note that the terminating 0 must be transferred as well. Combining several messages to one packet is not allowed. Length of the transferred string is not limited by this protocol.

Skype will not block after having received a message. If sending a message using SendMessage() fails this indicates that the communication channel is broken.

Note: Result of processing the message (both SkypeControlApiMessage and copydata) must be different from zero, otherwise Skype will consider the connection broken!

API client cannot spend more than 1 second processing the API messages, if a client spends more than 1 second for processing the connection will be disconnected. Use the PING command to test connection status. *In order to easen debugging while development one can enter the key APITimeoutDisabled (DWORD value, 0 = timeout enabled 1 = timeout disabled) into HKCU\Software\Skype\Phone\UI - this makes Skype not force the 1 second timeout.*

8.2 API Transport on Windows messages: Frequently asked questions

How long can Skype hold my SendMessage() call?

Skype does not hold the SendMessage() call at all (under normal circumstances.

How long should an application wait before resending a command?

Most commands should return within milliseconds, searched can take longer. Use PING every few seconds to make sure the communication channel is still alive.

Can I have multiple threads doing SendMessage()?

You can, but there is no real difference as windows itself forces these to go into one thread.

8.3 API Transport on Windows messages: deprecated

To initiate communication, the Client should broadcast window message ('SkypeControlAPI') to all windows in the system, specifying its own window handle in wParam parameter. In response to that Skype responds to the same message to the handle specified, and indicates its communication window handle in wParam. Now both parties know the window handle to send messages.

Client may continue polling Skype with 'SkypeControlAPI' message to check if it is still available.