Skype Desktop API Reference Manual

(Formerly known as Skype Public API)

Quick link to the full index of this reference.

Purpose of this guide

This document describes the Skype application programming interface (API) for Windows, the Skype APIs for Linux and Mac, and provides a reference guide for the Skype developer community.

Who reads this guide?

Skype's developer community who work with us to enrich the Skype experience and extend the reach of free telephone calls on the internet.

What is in this guide?

This document contains the following information:

- Overview of the Skype API
- Using the Skype API on Windows
- Using the Skype API on Linux
- Using the Skype API on Mac
- Skype protocol
- Skype reference
  - Terminology
  - Commands
  - Objects
  - Object properties
  - General parameters
  - Notifications
  - Error codes
- Skype URI
- Skype release notes

More information

- Share ideas and information on the Skype Desktop API forum on the Skype websites.

Desktop API (Accessories)

Overview
Terms of Use
API Reference
Certification
Forums

Tools & Downloads

Tracer
Skype4COM

Legal information

Text notation

This document uses `monospace` font to represent code, file names, commands, objects and parameters. The following text conventions apply for syntax:

- `CALL` – uppercase text indicates a keyword, such as command, notification, and object.
- `property` – lowercase text indicates a category of a keyword
- `<username>` – angle brackets indicate an identifier, such as username or call id
- `[<target>]` – square brackets identify optional items
- * – asterisk indicates repetitive items
- | – vertical bar means "or"
- `->` – command issued by client (used in examples)
- `<-` – response or notification from Skype (used in examples)
- `//` – comment line (used in examples)

Overview of the Skype API

The Skype API provides a mechanism for 3rd party scripts, applications and devices to control Skype UI functions and implement additional or improved features to complement the Skype.

The API has two layers:

- Communication Layer – is a set of methods for external application to establish connection to Skype client and communicate with it.
- Command Protocol Layer – is a text-based "language" that external applications can use to speak to the Skype client, once communication channel is established by Communication Layer.

Additionally, there are several Skype API wrapper libraries that encapsulate the functionality of Skype API. Such wrappers can act as optional third layers.

Communication Layer

Communication Layer provides a mechanism for external application to communicate with Skype. This layer is platform-dpendant – a transport mechanism to exchange data with Skype is different on Windows, Linux and Mac operating systems.

For more information on how to implement communication layers for different operating systems, see following sections of this document:

- Using the Skype API on Windows
- Using the Skype API on Linux
- Using the Skype API on Mac

Once your application has attached itself to Skype via Communication Layer, it can forget all about it and proceed with talking to Skype, using Protocol layer commands.

Protocol Layer

The Protocol Layer is a language of commands that Skype knows how to respond to. The syntax of that language is described in Skype API reference portion of this documument.

Commands sent to Skype must be in UTF-8 format.

To get a better feel how the command protocol works, you should start by downloading the Skype API Tracer program. Once you get that program running (and have authorised its connection to the API in Skype UI) you can play around with commands you can find in the Commands section.

For example, you can query various properties of a contact record (User object) like this:

```
-> get user echo123 birthday
<- USER echo123 BIRTHDAY 0
-> get user echo123 is_video_capable
<- USER echo123 IS_VIDEO_CAPABLE FALSE
```

A test call to Skype's call testing service using API would look approximately like that:

```
-> call echo123
<- CALL 14662 STATUS UNPLACED
<- CALL 14662 STATUS UNPLACED
<- CALL 14662 STATUS ROUTING
<- USER echo123 COUNTRY United Kingdom
<- USER echo123 COUNTRY United Kingdom
<- USER echo123 COUNTRY
<- CALL 14662 STATUS RINGING
<- USER echo123 COUNTRY United Kingdom
<- CALL 14662 VAA_INPUT_STATUS FALSE
<- CALL 14662 STATUS INPROGRESS
<- CALL 14662 DURATION 1
<- CALL 14662 DURATION 2
<- CALL 14662 DURATION 3
<- CALL 14662 STATUS FINISHED
```

Wrappers

While text based command protocol is more universal, using pre-built libraries is easier to start with. We have had three API wrapper libraries: Skype4COM, Skype4Py and Skype4Java. Currently, only Skype4COM wrapper is still supported.

Skype API on Windows

When developing applications to work with Skype, follow these general guidelines:

- Give intuitive names to executable files (.exe files) because this name is displayed to the user for confirmation. If the name is unclear, the user might not allow the application to access Skype.
- Sign applications with VeriSign's CodeSigning certificate.
- The application must support the `NAME` command and publish its name.
  Skype for Windows sends and receives API commands using `WM_COPYDATA` messages. Use the `RegisterWindowMessage` method to register the following messages:
- `SkypeControlAPIDiscover`
- `SkypeControlAPIAttach`
  To initiate communication, a client application broadcasts the `SkypeControlAPIDiscover` message, including its window handle as a `wParam` parameter. Skype responds with a `SkypeControlAPIAttach` message to the specified window and indicates the connection status with one of the following values:
- `SKYPECONTROLAPI_ATTACH_SUCCESS = 0` – The client is attached and the API window handle is provided in `wParam` parameter.
- `SKYPECONTROLAPI_ATTACH_PENDING_AUTHORIZATION = 1` – Skype acknowledges the connection request and is waiting for user confirmation. The client is not yet attached and must wait for the `SKYPECONTROLAPI_ATTACH_SUCCESS` message.
- `SKYPECONTROLAPI_ATTACH_REFUSED = 2` – The user has explicitly denied access to client.

- SKYPECONTROLAPI_ATTACH_NOT_AVAILABLE = 3 – The API is not available at the moment, for example because no user is currently logged in. The client must wait for a SKYPECONTROLAPI_ATTACH_API_AVAILABLE broadcast before attempting to connect again.
  When the API becomes available, Skype broadcasts the SKYPECONTROLAPI_ATTACH_API_AVAILABLE = 0x8001 message to all application windows in the system. The data exchange uses commands (or responses), provided as null-terminated UTF-8 strings. The terminating 0 must be transferred as well. You cannot combine several messages in one packet. There is no limit to the length of the transferred string.

Note: The result of processing the message must be different from zero (0), otherwise Skype considers that the connection broken.

If the API client spends more than 1 second processing a message, the connection is disconnected. Use the PING command to test the connection status. To ease debugging during development, in regedit enter the key APITimeoutDisabled (DWORD value, 0 = timeout enabled 1 = timeout disabled) into the HKCU\Software\Skype\Phone\UI file in the registry to override the 1 second timeout.

To check if Skype is installed, in regedit check if the following key exists: HKCU\Software\Skype\Phone, SkypePath . This key points to the location of the skype.exe file . If this key does not exist, check if the HKLM\Software\Skype\Phone, SkypePath key exists. If the HKCU key does not exist but the HKLM key is present, Skype has been installed from an administrator account but not been used from the current account.

Skype API on Linux

The Skype API for Linux, version 1.4 uses the Skype protocol 7, with few limitations in comparison to protocol 7 implementation in our Windows version. The list of unavailable commands can be found at the bottom of this page.

Supported distributions

Skype for Linux runs on the following Linux distributions:

- Feisty Fawn (7.04)
- Debian Etch
- Mepis
- Xandros
- Fedora 7 / Fedora Core 6
- OpenSUSE 10+
- Mandriva
- Dynamic / Static / Static OSS
  The client may also work with other distributions but has not been tested.

Transport

Use the Skype API for Linux, version 1.3, with either:

- D-BUS messaging
- X11 messaging

Note: X11 messaging is still under development. The final release of Skype for Linux API, version 1.3, will include examples of working with X11 and a description of the Skype action handler for X11.

X11 messaging

The X11 messaging framework is included in all Linux distributions.

D-BUS messaging

Download the D-BUS libraries, version 0.23

D-BUS behavior in this release is changed from earlier releases, as follows:

- D-BUS is disabled by default to avoid startup delays for developers who do not want to use it.
- To use D-BUS in a manner that is consistent with earlier versions of the Skype API for Linux, enter the following switches in the command line when you start the Skype client:
  `--enable-dbus --use-system-dbus`

The second switch is necessary because Skype now uses the session-dbus by default to enable multiple clients to run on one machine simultaneously.

Important: The Skype for Linux API, version 1.3, beta uses D-BUS version .23. The next release will move to support for D-BUS version .61+.

If you use RPM Package Manager to install skype, the D-BUS files are automatically configured. If you do not use RPM for the installation, you must create a configuration file as follows:

1. Create a text file named `skype.conf`
2. Save this file to `/etc/dbus-1/system.d/skype.conf`
3. Add the following information to the file:

```
<!DOCTYPE busconfig PUBLIC "-//freedesktop//
DTD D-BUS Bus Configuration 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>
<policy context="default">
<allow own="com.Skype.API"/>
<allow send_destination="com.Skype.API"/>
<allow receive_sender="com.Skype.API"/>
<allow send_path="/com/Skype"/>
</policy>
</busconfig>
```

Using the Skype API for Linux

To access the Skype API from a client application:

- The application passes its name to Skype:
  `-> NAME <application_name>`
- Skype pops up the following response to the user:<- wants to talk to Skype. OK?

  Important: On Linux, if you use spaces in the application_name, the name is truncated to the space. For example, if the application name is Skype for Java, the message displayed is "Skype wants to talk . . .". Also, on Linux it is essential to pass the application name before exchanging protocols, otherwise the connection will not work.

- If the user selects OK, protocol messages are exchanged:

```
<- OK
-> PROTOCOL 5
<- PROTOCOL 5
```

The Skype protocol manages the subsequent session.

Note: The session is associated with the window ID of the Skype API client. If the window is closed for any reason, a new session must be established.

D-BUS usage

D-BUS uses the following:

- Service is `com.Skype.API`
- Communication paths:
  - The client-to-Skype path is `/com/Skype`.
  - The Skype-to-client path is `/com/Skype/Client`.
- Methods are:
  - Use the `Invoke` method with one string parameter for client-to-Skype commands.
  - Use the `Notify` method for Skype-to-client commands and responses.
    D-BUS is disabled by default.

Protocol 7 commands currently missing from Linux version

- `GET / SET AVATAR`
- `GET / SET PCSPEAKER`
- `GET / SET RINGTONE`
- `GET / SET UI_LANGUAGE`
- `GET / SET VIDEO_IN`
- `GET / SET WALLPAPER`
- `GET / SET SILENT_MODE`
- `GET PREDICTIVE_DIALER_COUNTRY`
- `GET SKYPEVERSION`
- `GET USER`
- `SET MENU_ITEM`
- `RINGER`

Also most of the `OPEN` commands for various Skype UI windows have not been implemented yet.

Skype API on Mac

The Skype API is available in Skype for Mac OS X starting from version 1.3 and has interfaces for Cocoa, Carbon, and AppleScript. The Cocoa and Carbon interfaces are implemented in Skype.framework. Skype recommends that you include the Skype framework in your application as an embedded framework. To do so, copy it into the application bundle and link it to the application.

Client applications send string commands to control Skype. The format of these strings commands is described in the Skype API reference. If you are using a Cocoa or Carbon interface, Skype will send information back to your application by calling asynchronous delegate functions/methods.

Below, you'll find the instructions specific to Cocoa, Carbon, and AppleScript.

Cocoa

SkypeAPI class

Class methods

```
+ (BOOL)isSkypeRunning;
```

This method returns YES, when Skype is running and NO otherwise.

```
+ (void)setSkypeDelegate:(NSObject<SkypeAPIDelegate>*)aDelegate;
```

You must design an object to be Skype delegate (see delegate methods below). Use this method to set your object as Skype delegate.

```
+ (NSObject<SkypeAPIDelegate>*)skypeDelegate;
```

Returns the object which is currently set as Skype delegate.

```
+ (void)removeSkypeDelegate;
```

Removes current Skype delegate.

```
+ (void)connect;
```

Call this method after you have set Skype delegate. It will try to connect your application to Skype. Delegate method `skypeAttachResponse` will let you know, whether your application was successfully connected or not.

```
+ (void)disconnect;
```

Disconnects your application from Skype.

```
In 2.5 and later:
+ (NSString*)sendSkypeCommand:(NSString*)aCommandString;

In 1.5:
+ (void)sendSkypeCommand:(NSString*)aCommandString;
```

Use this method to control Skype or request information. `aCommandString` is a Skype API string as described in Skype API protocol documentation. Note, that if you are using Skype.framework 2.5 or later then you have to change your code a little bit compared to 1.5, because in 2.5 sendSkypeCommand returns strings (in 1.5 all information was returned in asynchronous callbacks).

Delegate methods

Required method

```
// delegate protocol
@protocol SkypeAPIDelegate
- (NSString*)clientApplicationName;
@end
```

This method should return the name of your application. This name will be shown to the user, when your application uses Skype. The name should not include any version information.

Optional methods

```
// delegate informal protocol
@interface NSObject (SkypeAPIDelegateInformalProtocol)
- (void)skypeNotificationReceived:(NSString*)aNotificationString;
```

This is the main delegate method Skype uses to send information to your application. `aNotificationString` is a Skype API string as described in Skype API protocol documentation.

```
- (void)skypeAttachResponse:(unsigned)aAttachResponseCode;
```

This method is called after Skype API client application has called connect. `aAttachResponseCode` is 0 on failure and 1 on success.

```
- (void)skypeBecameAvailable:(NSNotification*)aNotification;
```

This method is called after Skype has been launched.

```
- (void)skypeBecameUnavailable:(NSNotification*)aNotification;
```

This method is called after Skype has quit.

```
@end
```

Guidelines

Design an object in your application to be a Skype delegate. This object must implement the required delegate method `clientApplicationName`. In order to receive information from Skype, it is recommended that your delegate object also implements the optional delegate methods. The first method your application should call is `setSkypeDelegate`. In most implementations, that will probably be:

```
[SkypeAPI setSkypeDelegate:self];
```

Next, you should call connect. After you have received positive response with `skypeAttachResponse`, you can start sending commands to Skype by using `sendSkypeCommand`. For example:

```
[SkypeAPI sendSkypeCommand:@"CALL echo123"];
```

When your application quits or wants to disconnect from Skype, you should call disconnect.

Carbon

In order to use Skype API, you must create a single instance of struct `SkypeDelegate`. If you set callback functions for the members of this struct, then Skype will call these functions to send information to your application. The only required member of this struct is a string `clientApplicationName`.

Here is the definition of `SkypeDelegate`:

```
struct SkypeDelegate
{
// Required member
CFStringRef clientApplicationName;
// Optional members, can be NULL
void (*SkypeNotificationReceived)(CFStringRef aNotificationString);
void (*SkypeAttachResponse)(unsigned int aAttachResponseCode);
void (*SkypeBecameAvailable)(CFPropertyListRef aNotification);
void (*SkypeBecameUnavailable)(CFPropertyListRef aNotification);
};
```

Description

```
CFStringRef clientApplicationName;
```

This string should be the name of your application. It will be shown to the user, when your application uses Skype. The name should not include any version information.

```
void (*SkypeNotificationReceived)(CFStringRef aNotificationString);
```

This is the main delegate function Skype uses to send information to your application.

`aNotificationString` is a Skype API string as described in Skype API protocol documentation.

```
void (*SkypeAttachResponse)(unsigned int aAttachResponseCode);
```

This function is called after Skype API client application has called `ConnectToSkype`. `aAttachResponseCode` is 0 on failure and 1 on success.

```
void (*SkypeBecameAvailable)(CFPropertyListRef aNotification);
```

This function is called after Skype has been launched.

```
void (*SkypeBecameUnavailable)(CFPropertyListRef aNotification);
```

This function is called after Skype has quit.

You should define the functions like this:

```
void SkypeNotificationReceived(CFStringRef aNotificationString){}
void SkypeAttachResponse(unsigned int aAttachResponseCode){}
void SkypeBecameAvailable(CFPropertyListRef aNotification){}
void SkypeBecameUnavailable(CFPropertyListRef aNotification){}
```

and you can set them as members of your `SkypeDelegate` struct like so:

```
SkypeDelegate mySkypeDelegate;
mySkypeDelegate.SkypeNotificationReceived = SkypeNotificationReceived;
mySkypeDelegate.SkypeAttachResponse = SkypeAttachResponse;
mySkypeDelegate.SkypeBecameAvailable = SkypeBecameAvailable;
mySkypeDelegate.SkypeBecameUnavailable = SkypeBecameUnavailable;
mySkypeDelegate.clientApplicationName = CFSTR("My Carbon App");
```

Skype API methods

```
Boolean IsSkypeRunning(void);
```

This function returns TRUE, when Skype is running and FALSE otherwise.

```
void SetSkypeDelegate(struct SkypeDelegate* aDelegate);
```

You must design a struct to be Skype delegate (see `SkypeDelegate` description above). Use this function to set your struct as Skype delegate.

```
struct SkypeDelegate* GetSkypeDelegate(void);
```

Returns the struct which is currently set as Skype delegate.

```
void RemoveSkypeDelegate(void);
```

Removes current Skype delegate.

```
void ConnectToSkype(void);
```

Call this function after you have set Skype delegate. It will try to connect your application to Skype. Delegate callback function `skypeAttachResponse` will let you know, whether your application was successfully connected or not.

```
void DisconnectFromSkype(void);
```

Disconnects your application from Skype.

```
CFStringRef SendSkypeCommand(CFStringRef aCommandString);
```

Use this function to control Skype or request information. `aCommandString` is a Skype

API/span> string as described in Skype API protocol documentation.

In Skype.framework 2.6.0.142 and later: CFStringRef !SendSkypeCommand(CFStringRef aCommandString);
Older versions: void !SendSkypeCommand(CFStringRef aCommandString);

Note, that if you are using Skype.framework 2.6.0.142 or later then you have to change your code a little bit compared to older versions, because in 2.6.0.142 !SendSkypeCommand returns strings (previously all information was returned in asynchronous callbacks). Skype versions 2.5 and higher know how to return info synchronously. So, if you want to support Skype version 1.5, then you still have to listen to asynchronous callbacks.

Guidelines

The first method your application should call is `SetSkypeDelegate`, where `aDelegate` is your `SkypeDelegate` struct. In most implementations, that will probably be:

```
SetSkypeDelegate(&amp;myCarbonDelegate);
```

Next, you should call `ConnectToSkype`. After you have received positive response with `SkypeAttachResponse`, you can start sending commands to Skype by using `SendSkypeCommand`. For example:

```
SendSkypeCommand(CFSTR("CALL echo123"));
```

When your application quits or wants to disconnect from Skype, you should call `DisconnectFromSkype`.

AppleScript

There is just one command for Skype API, but it is a very powerful command, because you can send any the command strings as specified in Skype API protocol documentation to control Skype or request information.

Examples

```
tell application "Skype"
*send command "MESSAGE echo123 check" script name "My Script"
end tell
tell application "Skype"
*send command "CALL echo123" script name "My Other Script"
end tell
```

Skype protocol

The Skype protocol is currently in its seventh version. Starting with protocol 1 (the first Skype protocol) a new version is created only when new commands become incompatible with existing commands. The protocol number does not increase when new commands are introduced but existing commands remain unchanged.

Protocol 8

Protocol 8 is the current version of the Skype protocol.

- New CALL STATUS enumerator – `WAITING_REDIAL_COMMAND`.

- New CALL STATUS enumerator – REDIAL_PENDING.
- New SMS FAILUREREASON enumerator – NO_SENDERID_CAPABILITY.
- Sending chat messages and CHAT CREATE commands may now fail with a new error code: 615, "CHAT: chat with given contact is disabled".

Protocol 7

- Call transfer API, We have two new CALL statuses: TRANSFERRING|TRANSFERRED
- Modified CHATMESSAGE property TYPE enumerations:

```
TYPE = POSTEDCONTACTS|GAP_IN_CHAT|SETROLE|KICKED|SETOPTIONS|
KICKBANNED|JOINEDASAPPLICANT|SETPICTURE|SETGUIDELINES
```

Protocol 6

- VOICEMAIL command enters deprecation process and is replaced by CALLVOICEMAIL command.

Protocol 5

Protocol 5 is the current version of the Skype protocol and is used by the following versions of Skype:

- 2.0 – Windows
- 1.4.0.84 – Windows
- 1.3.0.33 – Windows and Mac
  This protocol introduced multiperson chat commands, one-to-one video calls, call forwarding, and contact grouping.

Protocol 4

Protocol 4 is used by the following versions of Skype:

- 1.2.0.11 – Windows
- 1.1.0.3 – Windows and Linux
  This protocol introduced ISO code prefixes for language and country.

Protocol 3

Protocol 3 is used by the following version of Skype:

- 1.1.0.61 – Windows
  This protocol introduced a compatibility layer for previous versions of instant messaging.

Protocol 2

Protocol 2 is used by the following version of Skype:

- 1.0.0.94
  This protocol implemented the following changes:
- Introduced the SKYPEME online status
- For calls on hold, notifies clients with either LOCALHOLD or REMOTEHOLD . Protocol 1 simply returned ONHOLD .

- Introduces the call status, `CANCELLED` .

Protocol 1 and 2 compatibility

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

- `SEARCH MESSAGES` → `SEARCH CHATMESSAGES`
- `SEARCH MISSEDMESSAGES` → `SEARCH MISSEDCHATMESSAGES`
- `GET MESSAGE` → `GET CHATMESSAGE`
- `SET MESSAGE` → `SET CHATMESSAGE`
  The `GET MESSAGE` properties are also converted:
- `PARTNER_HANDLE` → `FROM_HANDLE`
- `PARTNER_DISPNAME` → `FROM_DISPNAME`
  All API notification (including `GET/SET MESSAGE`) replies are converted:
- `CHATMESSAGE * FROM_HANDLE x` → `MESSAGE * PARTNER_HANDLE x`
- `CHATMESSAGE * FROM_DISPNAME x` → `MESSAGE * FROM_DISPNAME x`
- `CHATMESSAGE * property x` → `MESSAGE * property x`
  If the protocol is less than 3, `SEARCH MESSAGES` and `SEARCH MISSEDMESSAGES` commands return string `MESSAGES` 1, 2, 3.

Skype API reference

The Skype API reference is a guide for developers working with the Skype Desktop API.

Terminology

The Skype API reference uses the following terms:

- The Skype access API is also known as the Skype control API.
- The client application issues a **command** to control Skype.
- In reply to some commands, Skype returns a synchronous **response**. Not all commands require a response. Responses are documented under their relevant commands.
- Skype **objects** and their properties are described in Objects section of this reference.
- A **notification** is an asynchronous message Skype sends to a client when a change occurs, for example when a contact comes online or a new chatmessage is received.
- Skype has **general parameters** to control the setup, current user and connection information.
- **Connectable users** are online Skype users who are in the client contact list and also non-contacts who are in active communication with the client.

Commands

This section provides a reference to the commands used in Skype.

Command identifiers

A command identifier is useful to identify a response to a specific command. A command identifier is supported by most commands and is included in the response.

Syntax

```
#<command_id> command
```

Response

```
#<command_id> response|error
```

Parameters

`command_id` – client assigned alphanumeric identifier

Errors
all possible errors for a given command

Version
Protocol 4

Notes

- A command identifier is not included in asynchronous notification events initiated by a command.
- Asynchronous commands usually return a synchronous response with the command id. When the command is processed an asynchronous notification is also sent
- A response may come not directly after the command because there can other messages can be received between command and response.

Examples

Simple response to command

```
-> #AB GET USERSTATUS
<- #AB USERSTATUS ONLINE
```

Invalid command with reported error

```
-> #123 GET XZY
<- #123 ERROR 7 GET: invalid WHAT
```

Command response and notification

```
-> #cmd11 SET USERSTATUS ONLINE
// this is the response for the command
<- #cmd11 USERSTATUS ONLINE
// this is notification when the command is actually processed
<- USERSTATUS ONLINE
```

Command response and notification are asynchronous

```
-> #50 CALL +18005551234
// note that events can arrive before response
<- CALL 651 STATUS ROUTING
<- #50 CALL 651 STATUS ROUTING
<- CALL 651 PSTN_STATUS 10503 Service Unavailable
// the following events do not have a command id
<- CALL 651 FAILUREREASON 1
<- CALL 651 STATUS FAILED
```

Notifications can appear between command-response

```
-> #50 PING
// note that other events can arrive before command response
<- USER echo123 LASTONLINETIMESTAMP 1105764678
<- USER echo123 FULLNAME Echo Test Service
<- USER test LASTONLINETIMESTAMP 1105487965
// Now comes Skype response to command
<- #50 PONG
```

Making and managing voice calls

This section describes the commands for making and managing voice calls.

Refer to Making and managing video calls for a description of video call commands.
Refer to Call failure reasons for a list of all reasons for call failure.

CALL

Syntax

```
CALL <target>[, <target>]*
```

Response
```
CALL <call_ID> <status>
```

Parameters
`<target>` – targets to be called. In case of multiple targets conference is created. Available target types:

- `USERNAME` – Skype username, e.g. "pamela", "echo123"
- `PSTN` – PSTN phone number, e.g. "+18005551234", "003725555555"
- `SPEED DIAL CODE` – 1 or 2 character speeddial code

Errors

- `ERROR 34 invalid user handle`
  Target username/number missing or contains invalid characters
- `ERROR 39 user blocked`
  Trying to call to a blocked user (unblock user in contactlist)
- `ERROR 73 too many participants`
  Call is initiated to more than 9 people
- `ERROR 92 call error`
  Call is initiated to a number that is neither PSTN number nor a speeddial number

Version
Protocol 1

Notes
The Skype call window is focused when a call is initiated through the API. It is possible to make speed dial calls via the API.

Example

```
-> CALL echo123
<- CALL 1402 STATUS ROUTING
<- CALL 1402 SUBJECT
<- CALL 1402 STATUS ROUTING
<- CALL 1402 STATUS RINGING
<- CALL 1402 STATUS INPROGRESS
<- CALL 1402 DURATION 1
<- CALL 1402 STATUS FINISHED
```

GET CALL

Syntax
```
GET CALL <id> property
```

Response
```
CALL <id> property <value>
```

Parameters and response values

- `<id>` – call ID (numeric);
- `property` – property name. Refer to CALL object for the list of properties.

Errors

- `ERROR 7 GET: invalid WHAT`
  Object name missing or misspelled.
- `ERROR 11 Invalid call id`
  ID includes other than numeric characters.
- `ERROR 12 Unknown call`
  Call with specified ID does not exist in current user's call history.
- `ERROR 13 Invalid prop`
  Property name missing or misspelled.
- `ERROR 71 Invalid conference participant NO`
  Conference participant's number is not a number or is too big

Version
Protocol 1

Example

```
-> GET CALL 1594 TYPE
<- CALL 1594 TYPE OUTGOING_P2P
```

SET CALL INPROGRESS

This enables you to resume a call, for example after placing it on hold.

Syntax:
```
-> SET CALL <id> STATUS INPROGRESS

<- CALL <id> STATUS INPROGRESS
```

Parameters:
`<id>` – call ID (numeric)

Errors

- `ERROR 19 Invalid call id`
  ID includes other than numeric characters
- `ERROR 20 Unknown call`
  Call with specified ID does not exist
- `ERROR 23 Cannot resume this call at the moment`
  Given call is not ringing and therefore can not be answered.

SET CALL FINISHED

Terminates the call.

Syntax:
```
-> SET CALL <id> STATUS FINISHED

<- CALL <id> STATUS FINISHED
```

Parameters:
`<id>` – call ID (numeric)

Errors

- `ERROR 19 Invalid call id`
  ID includes other than numeric characters
- `ERROR 20 Unknown call`
  Call with specified ID does not exist in current user's call history nor is active.
- `ERROR 24 Cannot hangup inactive call`
  Given call is not in progress and therefore can not be hung up.

SET CALL ONHOLD

Places a call on hold. You can later resume the call by setting the state to `INPROGRESS`.

Syntax:
```
-> SET CALL <id> STATUS ONHOLD

<- CALL <id> STATUS ONHOLD
```

Parameters:

<id> – call ID (numeric), possible values:

Note that from Protocol 2 and up, SET CALL ONHOLD results in two possible status responses:

- LOCALHOLD – call was placed on hold by local user
- REMOTEHOLD – call was placed on hold by remote user

Errors

- ERROR 19 Invalid call id
  ID includes other than numeric characters
- ERROR 20 Unknown call
  The call ID does not exist in current user's call history nor is it active.
- ERROR 22 Cannot hold this call at the moment
  Given call is not in progress and therefore can not be placed on hold.
- ERROR 23 Cannot resume this call at the moment
  Given call is not on hold and therefore can not be resumed.

SET CALL JOIN CONFERENCE

Syntax
```
SET CALL <joining_id> JOIN_CONFERENCE <master_id>
```

Response
```
CALL <id> CONF_ID <conference_id>
```

Parameters

- <joining_id> – call ID (numeric) to join into;
- <master_id> – master call ID, where is another call's ID.

Errors

- ERROR 19 Invalid call id
  ID includes other than numeric characters
- ERROR 20 Unknown call
  Call with specified ID does not exist in current user's call history nor is active.
- ERROR 72 Cannot create conference
  Creating conference, for example " SET CALL 65 JOIN_CONFERENCE 66 " fails for some reason.

Note

- It is possible to initate a conference with the CALL target1, target2 command

Example

```
// make first call
-> CALL test
<- CALL 1540 STATUS ROUTING
<- CALL 1540 SUBJECT
<- CALL 1540 STATUS ROUTING
<- CALL 1540 STATUS RINGING
<- CALL 1540 STATUS INPROGRESS
// set first call on hold ...
-> SET CALL 1540 STATUS ONHOLD
<- CALL 1540 STATUS INPROGRESS
<- CALL 1540 STATUS ONHOLD
// .. and make another call
-> CALL echo123
<- CALL 1545 STATUS ROUTING
<- CALL 1545 SUBJECT
<- CALL 1545 STATUS ROUTING
<- CALL 1545 STATUS RINGING
<- CALL 1545 STATUS INPROGRESS
// join second call (1545) into conference with first call (1540)
-> SET CALL 1545 JOIN_CONFERENCE 1540
<- CALL 1545 CONF_ID 17930
```

```
<- CALL 1545 CONF_ID 17930
<- CALL 1540 CONF_ID 17930
// first call is automatically resumed and joined to conference
<- CALL 1540 STATUS INPROGRESS
// ...
<- CALL 1540 DURATION 53
<- CALL 1540 STATUS FINISHED
<- CALL 1545 DURATION 23
<- CALL 1545 STATUS FINISHED
```

SET CALL DTMF

Sends DTMF specified in parameter to the call target.

Syntax:

```
-> SET CALL <id> DTMF <value>

<- SET CALL <id> DTMF <value>
```

Parameters:

- `<id>` – call ID (numeric)
- `<value>` – permitted symbols are: {0..9,#,*}.

When sending DTMF codes manually, with the dialpad buttons on the Call Phones tab of the Skype UI, these DTMF codes are displayed on the address bar, below dialpad. This is not the case while sending DTMF codes with `SET CALL DTMF` command.

If you want your programmatically sent DTMF codes to be displayed on the address bar, you can use `BTN_RELEASED` command instead of `SET CALL DTMF`. When used during an active call, `BTN_RELEASED` with appropriate parameter {0..9,#,*} will cause equivalent DTMF code to be sent to the remote party of that call.

Note that this will only work if the Call Phones tab (dialpad) is active. On active Call tab, the DTMF codes will still be sent but the keys will not be displayed on the address bar. On Contacts tab, the keys will be added to the address bar but no DTMF codes will be sent. Therefore, if you want to use `BTN_RELEASED` for sending DTMF codes, you will need to make sure the Skype UI has Call Phones as active tab. This you can do with `OPEN DIALPAD` command.

Notes

- DTMF support and quality for PSTN calls depends on terminating partner.
- This command does not accept multiple symbols in its parameter.

Errors

- `ERROR 19 Invalid call id`
  ID includes other than numeric characters
- `ERROR 20 Unknown call`
  Call with specified ID does not exist in current user's call history nor is it active.
- `ERROR 21 Unknown/disallowed call prop`
  DTMF property value is incorrect or misspelled

SET CALL SEEN

Syntax

```
SET CALL <id> SEEN
```

Response

```
CALL <id> SEEN TRUE
```

Parameters

`<id>` – call ID (numeric)

Errors

- `ERROR 19 Invalid call id`
  ID includes other than numeric characters
- `ERROR 20 Unknown call`
  Call with specified ID does not exist in current user's call history nor is active.

Example

```
-> SET CALL 15 SEEN
<- CALL 15 SEEN TRUE
```

ALTER CALL

The `ALTER CALL` command controls call status.

Syntax:

```
ALTER CALL xxx
{ ANSWER
| HOLD
| RESUME
| HANGUP
| END { HANGUP | REDIRECT_TO_VOICEMAIL | FORWARD_CALL } // for an incoming ringing call
| DTMF <0|1|..|9|*|#>
| TRANSFER
| JOIN_CONFERENCE <callID> }
```

Refer to ALTER CALL TRANSFER command for more information on altering the `TRANSFER` property.

Command feedback for `ALTER CALL` always includes echoing back the original command, usually followed by status change notifications, specific to particular commands.

Example:

```
-> ALTER CALL 1719 HANGUP
<- ALTER CALL 1719 HANGUP
<- CALL 1719 STATUS FINISHED
```

Version
Protocol 5

GET CALL CAN_TRANSFER

Returns TRUE or FALSE, depending on whether a call can be transferred.

Syntax:
```
-> GET CALL <id> CAN_TRANSFER <handle>

<- CALL <id> CAN_TRANSFER <handle> {TRUE|FALSE}
```

Example:

```
-> GET CALL 1034 CAN_TRANSFER +3721234567
<- CALL 1034 CAN_TRANSFER +3721234567 FALSE
```

Version
Protocol 7 (API version 3.0)

ALTER CALL TRANSFER

Used for transferring an incoming call. Note that call transfers only work with incoming calls to SkypeIn numbers if you have Skype Pro subscription.

Syntax:

```
-> ALTER CALL <id> TRANSFER handle1[, handle2 ..]

<- ALTER CALL <id> TRANSFER
```

If multiple handles are passed in parameters, first one to answer the call will get the transfer.

To better describe the call transfer mechanism, let's assume there are three participants in a call: A, B and C.

- A calls B
- B transfers the call to C
- A and C can now talk.

The `ALTER CALL TRANSFER` command is issued by B, to create a call between A and C. To check whether it is possible to transfer the call from A, B can use GET CALL CAN_TRANSFER command. Note that it is caller B (transferring party) who has to determine, whether a call is transferable.

Relevant CALL object STATUS property values:

- `TRANSFERRING` – seen by B, this status is set while the call between A and C is in progress)
- `TRANSFERRED` – seen by B, terminating status of the call. Set after either the transferred call has ended or B does END/HANGUP;

Relevant CALL object properties:

- `TRANSFER_ACTIVE` – seen by A, indicates whether the call has been transferred.
- `TRANSFER_STATUS` – seen by B – the call status while the call is being transferred, it is relayed from A side continuously until the call has ended or when B decides to do CALL ALTER END. Ending call on B side will not terminate the call between A and C, just the status updates.
- `TRANSFERRED_BY` – seen by C, contains identity of B.
- `TRANSFERRED_TO` – seen by both A and B; contains identity of C.

Example:

```
//-------------------------------------------------------------------------------------------
// In this example, user Test is calling user Test3. Test3 then transfers the call to Test2.
// Note that for better clarity, call heartbeat messages are removed.
// Following portion of log is from perspective of the first outgoing call from user Test.
-> CALL Test3
<- CALL 626 STATUS UNPLACED
<- CALL 626 STATUS ROUTING
<- CALL 626 STATUS RINGING
<- CALL 626 TRANSFER_ACTIVE TRUE
<- CALL 626 STATUS ROUTING
<- CALL 626 TRANSFERRED_TO Test2
<- CALL 626 STATUS RINGING
<- CALL 626 VAA_INPUT_STATUS FALSE
<- CALL 626 STATUS INPROGRESS
<- CALL 626 VIDEO_STATUS VIDEO_NONE
<- CALL 626 STATUS FINISHED
//-------------------------------------------------------------------------------------------
// This portion of the log is from perspective of Test3 (who will transfer it to Test2)
<- CALL 288 CONF_ID 0
<- CALL 288 STATUS RINGING
<- CONTACTS FOCUSED
//-------------------------------------------------------------------------------------------
// Checking here if it is possible to transfer this call to Test2
-> GET CALL 288 CAN_TRANSFER Test2
<- CALL 288 CAN_TRANSFER test2 TRUE
//-------------------------------------------------------------------------------------------
// Transferring call to Test2
-> ALTER CALL 288 TRANSFER Test2
<- ALTER CALL 288 TRANSFER Test2
<- CALL 288 STATUS INPROGRESS
```

```
<- CALL 288 TRANSFERRED_TO Test2
<- CALL 288 TRANSFER_STATUS UNPLACED
<- CALL 288 TRANSFER_STATUS ROUTING
<- CALL 288 TRANSFER_STATUS RINGING
<- CALL 288 TRANSFER_STATUS INPROGRESS
<- CALL 288 STATUS FINISHED
<- CALL 288 VAA_INPUT_STATUS FALSE
//-------------------------------------------------------------------------------------------
// This portion of the log is from perspective of Test2 (receiver of the transferred call)
<- CALL 1218 CONF_ID 0
<- CALL 1218 STATUS RINGING
<- CONTACTS FOCUSED
-> ALTER CALL 1218 ANSWER
<- ALTER CALL 1218 ANSWER
<- CALL 1218 STATUS INPROGRESS
<- CALL 1218 VIDEO_STATUS VIDEO_NONE
<- CALL 1218 VAA_INPUT_STATUS FALSE
//-------------------------------------------------------------------------------------------
// Checking up who it was that transferred this call..
-> GET CALL 1240 TRANSFERRED_BY
<- CALL 1240 TRANSFERRED_BY Test3
<- CALL 1218 STATUS FINISHED
```

Version

Protocol 7 (API version 3.0)

Call failure reasons

Code	Description	Possible reason
1	CALL 181 FAILUREREASON 1	Miscellaneous error
2	CALL 181 FAILUREREASON 2	User or phone number does not exist. Check that a prefix is entered for the phone number, either in the form 003725555555 or +3725555555; the form 3725555555 is incorrect.
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.
13	CALL 181 FAILUREREASON 13	Failure to call a commercial contact.
14	CALL 181 FAILUREREASON 14	Conference call has been dropped by the host. Note that this does not normally indicate abnormal call termination. Call being dropped for all the participants when the conference host leavs the call is expected behaviour.

Sending and managing SMS messages

This section describes the commands for creating and managing SMS messages.

Refer to SMS object section for a list of SMS object properties.

CREATE SMS

This command creates an SMS object.

Syntax:
```
-> CREATE SMS <type> <target>
```

Where target is a valid PSTN number and type can be one of the following:

- `OUTGOING` – normal outbound SMS.
- `CONFIRMATION_CODE_REQUEST` – Refer to [#SMS_NUMBER_VALIDATION SMS reply-to validation] for more information.
- `CONFIRMATION_CODE_SUBMIT` – Refer to [#SMS_NUMBER_VALIDATION SMS reply-to validation] for more information.

Refer to

- Creating an SMS message section for more information (including format of feedback notifications).
- SMS object section for a list of SMS object properties.

Version
Added in API version 2.5

SET SMS BODY

This command sets or changes the text of an existing SMS object.

Syntax:
```
-> SET SMS <id> BODY "text"
```

Where is an SMS object ID returned from `CREATE SMS` command and text is the SMS message text.

Refer to

- Creating an SMS message section for more information.
- SMS object section for a list of SMS object properties.

Version
Added in API version 2.5

ALTER SMS SEND

This command sends a composed SMS message to the server.

Syntax:
```
-> ALTER SMS <id> SEND
```

Where is SMS object ID.

Refer to

- Creating an SMS message section for more information.
- SMS object section for a list of SMS object properties.

Version
Added in API version 2.5

SET SMS SEEN

This command sets an SMS object as SEEN.

Syntax:
```
-> SET SMS <id> SEEN
```

Where is an SMS object ID.

Refer to

- SMS object section for a list of SMS object properties.

Version
Added in API version 2.5

Creating an SMS message

To create, compose and send an SMS message, use `CREATE SMS`, `SET SMS` and `ALTER SMS` commands.

Refer to SMS object section for a list of SMS object properties.

Example:

```
// ---------------------------------------------------------------
// Here we create a new SMS object instance
-> CREATE SMS OUTGOING +0123456789
<- SMS 821 STATUS COMPOSING
<- SMS 821 PRICE 0
<- SMS 821 TIMESTAMP 0
<- SMS 821 PRICE_PRECISION 3
<- SMS 821 PRICE_CURRENCY EUR
<- SMS 821 STATUS COMPOSING
<- SMS 821 TARGET_NUMBERS +0123456789
<- SMS 821 PRICE -1
<- SMS 821 TARGET_STATUSES +0123456789=TARGET_ANALYZING
<- SMS 821 TARGET_STATUSES +0123456789=TARGET_ACCEPTABLE
<- SMS 821 PRICE 78
// ---------------------------------------------------------------
// This is how to set the message text property
// Note that you will get two identical lines in response
-> SET SMS 821 BODY "test 123 test 223 test 333"
<- SMS 821 BODY "test 123 test 223 test 333"
<- SMS 821 BODY "test 123 test 223 test 333"
// ---------------------------------------------------------------
// Now lets try to send the message
```

```
-> ALTER SMS 821 SEND
<- ALTER SMS 821 SEND
<- SMS 821 STATUS SENDING_TO_SERVER
<- SMS 821 TIMESTAMP 1174058095
<- SMS 821 TARGET_STATUSES +0123456789=TARGET_ACCEPTABLE
<- SMS 821 TARGET_STATUSES +0123456789=TARGET_DELIVERY_FAILED
<- SMS 821 FAILUREREASON INSUFFICIENT_FUNDS
<- SMS 821 STATUS FAILED
<- SMS 821 IS_FAILED_UNSEEN TRUE
// ---------------------------------------------------------------
// As sending the message failed (not enough Skype credit),
// lets delete the message
-> DELETE SMS 821
<- DELETE SMS 821
```

Version
Added in API version 2.5

SMS message text in chunks

The SMS object has special properties to break large messages into smaller chunks. Maximum size of a chunk is 160 characters. Note that some unusually clever-looking symbols (“ä”, “ö”, etc.) translate into more than one characters in stored text.

To query how many chunks is contained in an SMS message:

```
-> GET SMS <id> CHUNKING
<- SMS <id> CHUNKING <no. of chunks> <no. of characters in the final chunk>
```

To access text within a chunk:

```
-> GET SMS <id> CHUNK <#>
<- SMS <id> CHUNK <#> <text>
```

Searching SMS messages

Following two commands are available to search for SMS objects:

- SEARCH SMSS
- SEARCH MISSEDSMSS

Version
Added in API version 2.5

Deleting SMS messages

All SMS messages that you have created in Skype remain stored in the system until they get deleted. To delete an SMS message, use `DELETE SMS COMMAND`:

Syntax:

```
-> DELETE SMS <ID>

<- DELETE SMS <ID>
```

Example:

```
-> SEARCH SMSS
<- SMSS 233
-> DELETE SMS 233
<- DELETE SMS 233
```

The list of deletable SMS messages can be queried with SEARCH SMSS command. Refer to SMS object section for a list of SMS object properties.

Version
Added in API version 2.5

SET SMS REPLY_TO_NUMBER

This command sets the reply-to property of an SMS object.

Syntax:
```
-> SET SMS <id> REPLY_TO_NUMBER <pstn>
```

Version
Added in API version 2.5

SET SMS TARGET_NUMBERS

This command changes the destination(s) of an SMS message.

Syntax:
```
SET SMS <id> TARGET_NUMBERS <pstn1>[, <pstn2>]
```

Where is ID of a created SMS object and destination(s) are given as a comma-separated list of valid PSTN numbers.

Example:

```
//-----------------------------------------------------------------
// Note that at least one target number is mandatory for CREATE SMS
-> CREATE SMS OUTGOING +37259877305
<- SMS 1702 TYPE OUTGOING
<- SMS 1702 STATUS COMPOSING
<- SMS 1702 PRICE 0
<- SMS 1702 TIMESTAMP 0
<- SMS 1702 STATUS COMPOSING
<- SMS 1702 PRICE_PRECISION 3
<- SMS 1702 PRICE_CURRENCY EUR
<- SMS 1702 TARGET_NUMBERS +37259877305
<- SMS 1702 PRICE -1
<- SMS 1702 TARGET_STATUSES +37259877305=TARGET_ANALYZING
<- SMS 1702 TARGET_STATUSES +37259877305=TARGET_ACCEPTABLE
<- SMS 1702 PRICE 78
//-----------------------------------------------------------------
// Now let's add two more target numbers (in addition to original)
-> SET SMS 1702 TARGET_NUMBERS +37259877305, +37259877306, +37259877307
<- SMS 1702 TARGET_NUMBERS +37259877305, +37259877306, +37259877307
<- SMS 1702 TARGET_NUMBERS +37259877305, +37259877306, +37259877307
<- SMS 1702 PRICE -1
<- SMS 1702 TARGET_STATUSES +37259877305=TARGET_ACCEPTABLE, +37259877306=TARGET_ANALYZING,
+37259877307=TARGET_ANALYZING
<- SMS 1702 TARGET_STATUSES +37259877305=TARGET_ACCEPTABLE, +37259877306=TARGET_ACCEPTABLE,
+37259877307=TARGET_ACCEPTABLE
<- SMS 1702 TARGET_STATUSES +37259877305=TARGET_ACCEPTABLE, +37259877306=TARGET_ACCEPTABLE,
+37259877307=TARGET_ACCEPTABLE
<- SMS 1702 PRICE 234
```

Version
Added in API version 2.5

Setting mobile phone number on reply-to field in outgoing SMS messages

An outgoing SMS message from Skype lists the reply-to number as the user's Skype ID.
It is possible to change the reply-to number to a mobile phone number by registering the number in Skype client.
Skype validates this number, and it then becomes the reply-to number for outgoing SMS messages.

To register a mobile phone number in Skype client:

1. Create and send an SMS message of type `CONFIRMATION_CODE_REQUEST` to your own mobile number.
2. Skype sends an SMS message to your mobile, with message body containing a confirmation code.
3. Create another SMS of type `CONFIRMATION_CODE_SUBMIT` to the same number and include the confirmation code in message body.
4. Your mobile phone number is then validated as a reply-to number for outgoing SMS messages.

To create confirmation request and submit messages, use `CONFIRMATION_CODE_REQUEST` and `CONFIRMATION_CODE_SUBMIT` respectively as 2nd parameter in CREATE SMS command.

To retrieve the mobile number you have set as reply-to for outgoing SMS messages:

```
-> GET PROFILE SMS_VALIDATED_NUMBERS
<- PROFILE SMS_VALIDATED_NUMBERS <+ number >[, <+number>..]
```

Call cost information

Cost information is stored in `RATE`, `RATE_CURRENCY` and `RATE_PRECISION` properties of a `CALL` object.

Example of how to retrieve call cost data:

```
//-----------------------------------------------
// First let's find a suitable call ID
-> SEARCH CALLS
<- CALLS 100, 101, 102
//-----------------------------------------------
// Here we will retrieve cost data from call 100
-> GET CALL 100 RATE
<- CALL 100 RATE 1234
-> GET CALL 100 RATE_PRECISION
<- CALL 100 RATE_PRECISION 2
-> GET CALL 100 RATE_CURRENCY
<- CALL 100 RATE_CURRENCY EUR
//-----------------------------------------------
// To determine the actual cost of the call,
// you will also need to know the call duration
-> GET CALL 100 DURATION
<- CALL 100 DURATION 60
```

Note that call `DURATION` is expressed in seconds while call `RATE` is expressed as cost per minute.

Skype4Com example:

- CallCost.pas

Version

Protocol 6, Skype API version 2.5

Making and managing video calls

This section contains the commands for making and managing video calls.

Skype4Com sample:

- VideoSwitching.pas

GET VIDEO_IN

The `GET VIDEO_IN` command retrieves the name of the video device to use for a call. If no value is returned, Skype sets the default value.

Syntax
```
-> GET VIDE0_IN

<- VIDEO_IN [<devicename>]
```

Note
If no devicename is returned, Skype sets a default value with the following command:
```
-> SET VIDEO_IN <devicename>
```

SET VIDEO_IN

This command enables you to change webcam settings.

Syntax:
```
-> SET VIDE0_IN [<device_name>]

<- VIDEO_IN [<device_name>]
```

If the parameter is empty, webcam is set to “Default video device”.

If device passed in parameter cannot be found, following error is reported:

- ERROR 50 cannot set device

GET CALL VIDEO_STATUS

To check if a Skype client is video-enabled:

Syntax
```
-> GET CALL 5921 VIDEO_STATUS
```

Response

```
Skype responds with the video status for the active call, for example:
<- CALL 5921 VIDEO_STATUS VIDEO_NONE
```

Parameters

```
VIDEO_NONE
VIDEO_SEND_ENABLED
VIDEO_RECV_ENABLED
VIDEO_BOTH_ENABLED
```

Version
Protocol 5

ALTER CALL VIDEO_SEND

Used to start or stop sending video during a call.

Syntax to start video:
```
-> ALTER CALL <id> START_VIDEO_SEND

<- ALTER CALL <id> START_VIDEO_SEND

<- CALL <id> VIDEO_SEND_STATUS STARTING
```

Syntax to stop video:
```
-> ALTER CALL <id> STOP_VIDEO_SEND

<- ALTER CALL <id> STOP_VIDEO_SEND
```

Parameters:

START_VIDEO_SEND
STOP_VIDEO_SEND

Version
Protocol 5

ALTER CALL VIDEO_RECEIVE

Used to start or stop receiving video during a call.

Syntax to start receiving video:

```
-> ALTER CALL <id> START_VIDEO_RECEIVE

<- ALTER CALL <id> START_VIDEO_RECEIVE
```

Syntax to stop receiving video:

```
-> ALTER CALL <id> STOP_VIDEO_RECEIVE

<- ALTER CALL <id> STOP_VIDEO_RECEIVE

<- CALL <id> VIDEO_RECEIVE_STATUS STOPPING
```

Parameters:

START_VIDEO_RECEIVE
STOP_VIDEO_RECEIVE

Version
Protocol 5

GET CALL VIDEO_SEND_STATUS

To check video send status:

Syntax

```
-> GET CALL 5921 VIDEO_SEND_STATUS
```

Response
Skype responds with the appropriate parameter.

Parameters
NOT_AVAILABLE // The client does not have video capability because video is disabled or a webcam is unplugged).
AVAILABLE // The client is video-capable but the video is not running (can occur during a manual send).
STARTING // The video is sending but is not yet running at full speed.
REJECTED // The receiver rejects the video feed (can occur during a manual receive).
RUNNING // The video is actively running.
STOPPING // The active video is in the process of stopping but has not halted yet.
PAUSED // The video call is placed on hold.

Version
Protocol 5

GET CALL VIDEO_RECEIVE_STATUS

To check video receive status:

Syntax
```
-> GET CALL 5921 VIDEO_RECEIVE_STATUS
```

Response
Skype responds with the appropriate parameter.

Parameters
`NOT_AVAILABLE` // The client does not have video capability because video is disabled or a webcam is unplugged).
`AVAILABLE` // The client is video-capable but the video is not running (can occur during a manual send).
`STARTING` // The video is sending but is not yet running at full speed.
`REJECTED` // The receiver rejects the video feed (can occur during a manual receive).
`RUNNING` // The video is actively running.
`STOPPING` // The active video is in the process of stopping but has not halted yet.
`PAUSED` // The video call is placed on hold.

Version
Protocol 5

IS_VIDEO_CAPABLE

To check if a user is video-capable:

Syntax
```
-> GET USER <username> IS_VIDEO_CAPABLE
```

Response
```
<- USER <username> IS_VIDEO_CAPABLE {True|False}
```

Version
Protocol 5

OPEN VIDEOTEST

To open the Video Test window to test if video is working:

Syntax
```
OPEN VIDEOTEST
```

Response
If successful command is echoed back

Version
Protocol 5

OPEN OPTIONS VIDEO

To open the Video Options window:

Syntax:
```
-> OPEN OPTIONS VIDEO

<- OPEN OPTIONS VIDEO
```

Version
Protocol 5

Leaving and manipulating voicemails

This section contains the commands to leave and manipulate voicemails.

Skype4Com samples:

- VoiceMail2WAV.pas – Delphi example on how to save voicemails into WAV files.

VOICEMAIL

The `VOICEMAIL` command starts to deprecate in protocol 6 and is replaced by the CALLVOICEMAIL command.

CALLVOICEMAIL

Refer to VOICEMAIL object.

To leave a voicemail:

Syntax
```
CALLVOICEMAIL <target>
```

When you start an outgoing voicemail, a call object and two voicemail objects are created. First one of the voicemail objects is incoming greeting message. Second voicemail object is the outgoing message.

Example

```
//-------------------------------------------------------------------------------
// Starting voicemail call to testuser, the system will report back call
// ID and status. The object IDs in this example are call (524), greeting (525)
// and voicemail message (526)
-> CALLVOICEMAIL testuser
<- CALL 524 STATUS ROUTING
//-------------------------------------------------------------------------------
// Then the system reports back the incoming greeting voicemail properties
<- VOICEMAIL 525 TYPE CUSTOM_GREETING
<- VOICEMAIL 525 PARTNER_HANDLE testuser
<- VOICEMAIL 525 PARTNER_DISPNAME Test User
<- VOICEMAIL 525 ALLOWED_DURATION 60
<- VOICEMAIL 525 SUBJECT
<- VOICEMAIL 525 TIMESTAMP 1174384114
<- VOICEMAIL 525 DURATION 0
<- VOICEMAIL 525 STATUS NOTDOWNLOADED
<- VOICEMAIL 525 STATUS DOWNLOADING
//-------------------------------------------------------------------------------
// Then the system reports back the outgoing voicemail properties
<- VOICEMAIL 526 TYPE OUTGOING
<- VOICEMAIL 526 PARTNER_HANDLE testuser
<- VOICEMAIL 526 PARTNER_DISPNAME Test User
<- VOICEMAIL 526 ALLOWED_DURATION 600
<- VOICEMAIL 526 SUBJECT
<- VOICEMAIL 526 TIMESTAMP 1174384114
<- VOICEMAIL 526 DURATION 0
<- VOICEMAIL 526 STATUS BLANK
//-------------------------------------------------------------------------------
// The status of the call object is set to INPROGRESS, incoming greeting
// is being downloaded
<- CALL 524 STATUS INPROGRESS
<- CALL 524 VM_ALLOWED_DURATION 600
<- CALL 524 VM_DURATION 0
<- VOICEMAIL 525 STATUS PLAYING
<- VOICEMAIL 525 STATUS BUFFERING
<- CALL 524 STATUS INPROGRESS
<- VOICEMAIL 525 DURATION 8
//-------------------------------------------------------------------------------
// Incoming greeting has been received and is played
<- VOICEMAIL 525 TIMESTAMP 1125749735
<- VOICEMAIL 525 STATUS PLAYING
```

```
<- VOICEMAIL 525 STATUS PLAYED
//-------------------------------------------------------------------------------
// System starts recording the outgoing voicemail message
<- VOICEMAIL 526 TIMESTAMP 1174384125
<- VOICEMAIL 526 STATUS RECORDING
//-------------------------------------------------------------------------------
// Heartbeat notifications continue at 1 second interval throughout recording
<- CALL 524 STATUS INPROGRESS
<- VOICEMAIL 526 DURATION 6
<- CALL 524 VM_DURATION 6
<- VOICEMAIL 526 DURATION 7
<- CALL 524 VM_DURATION 7
<- VOICEMAIL 526 DURATION 8
<- CALL 524 VM_DURATION 8
<- VOICEMAIL 526 DURATION 9
<- CALL 524 VM_DURATION 9
//-------------------------------------------------------------------------------
// Recording stopped, uploading the recorded message
<- VOICEMAIL 526 STATUS UPLOADING
<- CALL 524 STATUS INPROGRESS
<- VOICEMAIL 526 STATUS UPLOADED
<- CALL 524 STATUS FINISHED
<- CALL 524 VAA_INPUT_STATUS FALSE
```

Version
Protocol 6

Prior to API version 2.5 (protocol 6), `VOICEMAIL` command was used to leave voicemails. In future development, `CALLVOICEMAIL` command should be used instead.
Also, following changes were made to this command in API version 2.5:

- When you create `VOICEMAIL` object, a `CALL` object is also created.
- After you play a voicemail, other person's greeting is not deleted.
- When voicemail is recording, Skype returns a call xx vm_duration x response in addition to voicemail xx duration x message.

Notes

- Leaving a voicemail for a target user actually uses two types of voicemail object:
  - a greeting type of voicemail object which is downloaded from the server
  - an outgoing type of voicemail object which the user composes

OPEN VOICEMAIL

To open and start playing a voicemail:

Syntax
```
OPEN VOICEMAIL <id>
```

Response
If successful command is echoed back

Parameters
`<id>` – voicemail identifier

Errors

- `ERROR 69 invalid open what`
  Open target is missing or misspelled
- `ERROR 512 invalid voicemail ID`
  Voicemail identifier is missing, is invalid or does not exist

Notes

- Voicemail is downloaded from server automatically.
- The main Skype window comes into focus and switches to the Call List tab; use the ALTER VOICEMAIL command to play without a UI response.

To get hold of voicemail IDs, refer to SEARCH VOICEMAILS and SEARCH MISSEDVOICEMAILS commands.

ALTER VOICEMAIL

The `ALTER VOICEMAIL` command allows finer control over the `VOICEMAIL` object.

Syntax:

```
-> ALTER VOICEMAIL <id> action

<- ALTER VOICEMAIL <id> action
```

Parameters:

`action` – possible values:

- `STARTPLAYBACK` – starts playing downloaded voicemail
- `STOPPLAYBACK` – stops voicemail playback
- `UPLOAD` – uploads recorded voicemail from a local computer to a server
- `DOWNLOAD` – downloads voicemail object from a server to a local computer
- `STARTRECORDING` – stops playing greeting and starts recording, the equivalent to a user pressing the green button;
- `STOPRECORDING` – ends recording, the equivalent to a user pressing the red button
- `DELETE` – delete voicemail object
- `STARTPLAYBACKINCALL` – Initiates voicemail playback during an active call. The voicemail will be played both locally and to remote call participant.
- `SETUNPLAYED` – sets voicemail status property to `UNPLAYED`.

In version 3.5.0.202 following `ALTER` commands were added to enable redirection of voice streams for voicemails:

- ALTER VOICEMAIL SET_INPUT
- ALTER VOICEMAIL SET_OUTPUT
- ALTER VOICEMAIL SET_CAPTURE_MIC

Notes

- `STARTPLAYBACK` plays voicemail but the window does not change to the Call List tab as it does with the `OPEN VOICEMAIL` command.
- `STOPRECORDING` causes automatic message upload to the server.
- Voicemails are deleted as a background process and the elapsed time depends on the server response; during this period, the `SEARCH VOICEMAILS` command still returns an ID for the voicemail, but the status is changed to `DELETING` .

Managing call forwarding

This section contains the commands to manage call forwarding.

Skype4Com example:

- CallForwarding.pas

GET PROFILE CALL_APPLY_CF

Use the `GET PROFILE CALL_APPLY_CF` command to query if call forwarding is enabled for a call.

Syntax

```
-> GET PROFILE CALL_APPLY_CF
```

Response

```
<- PROFILE CALL_APPLY_CF {True|False}
```

Version

Protocol 1.4

SET PROFILE CALL_APPLY_CF

Use the SET PROFILE CALL_APPLY_CF to enable or disable call forwarding.

Syntax

```
-> SET PROFILE CALL_APPLY_CF {True|False}
```

Response

```
<- PROFILE CALL_APPLY_CF {True|False}
```

Version

Protocol 1.4

GET PROFILE CALL_FORWARD_RULES

Use the GET PROFILE CALL_FORWARD_RULES to query the rules set for call forwarding. Note that the call forwarding process starts after number of seconds given in CALL_NOANSWER_TIMEOUT property of the PROFILE object.

Syntax:

```
-> GET PROFILE CALL_FORWARD_RULES

<- PROFILE CALL_FORWARD_RULES [<start_time>,<end_time>,{<username>|
<+PSTN>}[ <start_time>,<end_time>,{<username>|<+PSTN>}]*]
```

Parameters:

- start_time – in seconds when connecting to this number/user starts
- end_time – in seconds when ringing to this number/user ends
- username – another Skype username to forward calls to
- +PSTN - PSTN number to forward a call

Note

A call can be forwarded to multiple numbers and the numbers can overlap in time, with all ringing and the first to pick up the call takes it.

Version

Protocol 1.4

SET PROFILE CALL_FORWARD_RULES

Use the SET PROFILE CALL_FORWARD_RULES to set the rules for call forwarding. Note that the call forwarding process starts after number of seconds given in CALL_NOANSWER_TIMEOUT property of the PROFILE object.

Syntax:

```
-> SET PROFILE CALL_FORWARD_RULES [<start_time>,<end_time>,
{<username>|<+PSTN>}[ <start_time>,<end_time>,{<username>|
<+PSTN>}]*]

<- PROFILE CALL_FORWARD_RULES [<start_time>,<end_time>,{<username>|
<+PSTN>}[ <start_time>,<end_time>,{<username>|<+PSTN>}]*]
```

Parameters:

- start_time – in seconds when connecting to this number/user starts
- end_time – in seconds when ringing to this number/user ends
- username – another Skype username to forward calls to
- +PSTN - PSTN number to forward a call

Version
Protocol 1.4

GET PROFILE CALL_NOANSWER_TIMEOUT

Use the `GET PROFILE CALL_NOANSWER_TIMEOUT` to query the amount of seconds a forwarded call will ring before timing out.

Syntax
```
-> GET PROFILE CALL_NOANSWER_TIMEOUT
```

Response
```
<- PROFILE CALL_NOANSWER_TIMEOUT 15
```

Note

1. seconds is the default timeout value.

Version
Protocol 1.4

SET PROFILE CALL_NOANSWER_TIMEOUT

Use the `SET PROFILE CALL_NOANSWER_TIMEOUT` to change the amount of seconds a forwarded call will ring before timing out.

Syntax
```
-> SET PROFILE CALL_NOANSWER_TIMEOUT 20
```

Response
```
<- PROFILE CALL_NOANSWER_TIMEOUT 20
```

Note
This command replaces the default timeout value of 15 seconds.

Version
Protocol 1.4

GET PROFILE CALL_SEND_TO_VM

Use the `GET PROFILE CALL_SEND_TO_VM` to query if voicemail is enabled for forwarded calls.

Syntax
```
-> GET PROFILE CALL_SEND_TO_VM
```

Response
```
<- PROFILE CALL_SEND_TO_VM {True|False}
```

Version
Protocol 1.4

SET PROFILE CALL_SEND_TO_VM

Use the `SET PROFILE CALL_SEND_TO_VM` to enable (or disable) voicemail for forwarded calls.

Syntax
```
-> SET PROFILE CALL_SEND_TO_VM True
```

Response
```
<- PROFILE CALL_SEND_TO_VM True
```

Version
Protocol 1.4

Creating chats and sending messages

This section contains the commands for creating chats and sending messages.

CHAT CREATE

This command creates a chat object.

Syntax
```
-> CHAT CREATE [<target>, <target>*]
```

Response
```
<- CHAT <chat_id> STATUS <value>
```

Version
Protocol 5, updated in protocol 7 (API version 3.0)

Parameters

- `<target>` – username(s) with whom to create a chat
- `<chat_id>` – chat identifier; string (usually looks like "#me/$target;012345679012345")
- `<value>` – depends on the type of chat created: `DIALOG` for a 1:1 chat; `MULTI_SUBSCRIBED` for a chat with multiple participants

Notes

- From version 3.6 and later, opening chat windows (both from API and manually via UI) generate additional chat window open and close notfication messages. Refer to the Chat notifications section for more information.
- The `CHAT CREATE` command does not open a chat window; use the OPEN CHAT command to do so.
- Starting from protocol 7, the parameter(s) are no longer mandatory. If no usernames are passed in parameters, an empty multichat is created.

Example:

```
//-----------------------------------------------------------------
// Creating chat with one target
-> CHAT CREATE anappo5
<- CHAT #anappo/$anappo5;2e4e763a2fc121ed STATUS DIALOG
-> OPEN CHAT #anappo/$anappo5;2e4e763a2fc121ed
<- OPEN CHAT #anappo/$anappo5;2e4e763a2fc121ed
//-----------------------------------------------------------------
// Creating chat with no target
-> CHAT CREATE
<- CHAT #anappo/$72cb4c9d0871e6dc NAME #anappo/$72cb4c9d0871e6dc
<- CHAT #anappo/$72cb4c9d0871e6dc ACTIVITY_TIMESTAMP 0
<- CHAT #anappo/$72cb4c9d0871e6dc STATUS MULTI_SUBSCRIBED
<- CHAT #anappo/$72cb4c9d0871e6dc TYPE MULTICHAT
<- CHAT #anappo/$72cb4c9d0871e6dc STATUS UNSUBSCRIBED
<- CHATMEMBER 570 ROLE USER
<- CHAT #anappo/$72cb4c9d0871e6dc MYROLE USER
<- CHAT #anappo/$72cb4c9d0871e6dc MEMBERS anappo
<- CHAT #anappo/$72cb4c9d0871e6dc ACTIVEMEMBERS anappo
<- CHAT #anappo/$72cb4c9d0871e6dc MYSTATUS SUBSCRIBED
<- CHAT #anappo/$72cb4c9d0871e6dc STATUS MULTI_SUBSCRIBED
<- CHAT #anappo/$72cb4c9d0871e6dc TIMESTAMP 1175089677
-> OPEN CHAT #anappo/$72cb4c9d0871e6dc
<- OPEN CHAT #anappo/$72cb4c9d0871e6dc
//-----------------------------------------------------------------
// Creating chat with two targets
```

```
-> CHAT CREATE anappo3, anappo5
<- CHAT #anappo/$8c9e3bb94643d668 NAME #anappo/$8c9e3bb94643d668
<- CHAT #anappo/$8c9e3bb94643d668 ACTIVITY_TIMESTAMP 0
<- CHAT #anappo/$8c9e3bb94643d668 STATUS MULTI_SUBSCRIBED
<- CHAT #anappo/$8c9e3bb94643d668 TYPE MULTICHAT
<- CHAT #anappo/$8c9e3bb94643d668 STATUS UNSUBSCRIBED
<- CHATMEMBER 585 ROLE USER
<- CHAT #anappo/$8c9e3bb94643d668 MYROLE USER
<- CHAT #anappo/$8c9e3bb94643d668 MEMBERS anappo
<- CHAT #anappo/$8c9e3bb94643d668 ACTIVEMEMBERS anappo
<- CHAT #anappo/$8c9e3bb94643d668 MYSTATUS SUBSCRIBED
<- CHAT #anappo/$8c9e3bb94643d668 STATUS MULTI_SUBSCRIBED
<- CHAT #anappo/$8c9e3bb94643d668 TIMESTAMP 1175089858
<- CHAT #anappo/$8c9e3bb94643d668 MEMBERS anappo anappo3 anappo5
<- CHAT #anappo/$8c9e3bb94643d668 FRIENDLYNAME anappo3, anappo5
-> OPEN CHAT #anappo/$8c9e3bb94643d668
<- OPEN CHAT #anappo/$8c9e3bb94643d668
```

Error codes:

615, "CHAT: chat with given contact is disabled" – added in Skype version 3.5 (protocol 8)

CHATMESSAGE

Syntax
`CHATMESSAGE <chat_id> <message>`

Response
`CHATMESSAGE <id> STATUS SENDING`

Parameters

- `<chat_id>` – chat identifier
- `<message>` – message text body to send
- `<id>` – chatmessage identifier

Version
Protocol 5

Errors

- `ERROR 510 Invalid/unknown chat name given`
  Chat with does not exist
- `ERROR 511 Sending a message to chat fails`
  Could not send message to chat (eg. not a member)

ALTER CHAT SETTOPIC

Changes chat topic.

Syntax:
```
-> ALTER CHAT <chat_id> SETTOPIC <topic>

<- ALTER CHAT SETTOPIC
```

See also ALTER CHAT SETTOPICXML command.

Version
Protocol 5

Errors

- `ERROR 501 CHAT: No chat found for given chat`
  Chat with does not exist

ALTER CHAT SETTOPICXML

Enables you to set a chat topic that contains XML formatting elements. Note that the standard chat topic will be updated as well, stripped of XML tags.

Syntax:

```
-> ALTER CHAT <chat_id> SETTOPICXML <topic>

<- ALTER CHAT SETTOPICXML
```

Example (without feedback notifications):

```
-> ALTER CHAT #test/$b9275b3b334341f2 SETTOPICXML <BLINK>topic is blinking</BLINK>
-> ALTER CHAT #test/$b9275b3b334341f2 SETTOPICXML <B>topic in bold</B>
-> ALTER CHAT #test/$b9275b3b334341f2 SETTOPICXML <I>topic in italic</I>
-> ALTER CHAT #test/$b9275b3b334341f2 SETTOPICXML <U>topic with underline</U>
-> ALTER CHAT #test/$b9275b3b334341f2 SETTOPICXML Smiley: <SS type="smile">:-)</SS>
-> ALTER CHAT #test/$b9275b3b334341f2 SETTOPICXML <FONT COLOR="#FF0010">topic in red</FONT>
```

Version
Protocol 7 (API version 3.0)

ALTER CHAT ADDMEMBERS

This command adds new members to a chat.

Syntax:

```
-> ALTER CHAT <chat_id> ADDMEMBERS <target>[, <target>]*

<- ALTER CHAT ADDMEMBERS
```

Version
Protocol 5

Errors

- `ERROR 501 CHAT: No chat found for given chat`
  Chat with does not exist
- `ERROR 504 CHAT: Action failed`
  Could not add members into chat (eg is already a member; you have left chat)

ALTER CHAT LEAVE

This command causes user to leave the chat.

Syntax:

```
-> ALTER CHAT <chat_id> LEAVE

<- ALTER CHAT LEAVE
```

Errors

- `ERROR 501 CHAT: No chat found for given chat`
  Chat with does not exist
- `ERROR 504 CHAT: Action failed`
  Could not leave chat (for example if the user has already left this chat)

ALTER CHAT BOOKMARKED

Adds chat to the list of bookmarked chats.

Syntax to bookmark a chat:

```
-> ALTER CHAT <chat_id> BOOKMARK

<- ALTER CHAT <ID> BOOKMARKED TRUE
```

Syntax to remove a chat from list of bookmarked chats:

```
-> ALTER CHAT <ID> UNBOOKMARK

<- ALTER CHAT <ID> BOOKMARKED FALSE
```

Refer to following SEARCH commands on how to obtain a chat ID

SEARCH CHATS

SEARCH ACTIVECHATS

SEARCH MISSEDCHATS

SEARCH RECENTCHATS

SEARCH BOOKMARKEDCHATS

Version
Protocol 6, Skype API version 2.5

GET CHAT CHATMESSAGES

Returns IDs of chatmessage objects in a specified chat.

Syntax:

```
-> GET CHAT <chat_id> CHATMESSAGES

<- CHAT <chat_id> CHATMESSAGES <id>[, <id>]*
```

Version:
Protocol 5

Errors

- `ERROR 501 CHAT: No chat found for given chat`
  Chat with does not exist

GET CHAT RECENTCHATMESSAGES

Syntax

```
GET CHAT <chat_id> RECENTCHATMESSAGES
```

Response

```
CHAT <chat_id> RECENTCHATMESSAGES <id>[, <id>]*
```

Version
Protocol 5

Errors

- `ERROR 501 CHAT: No chat found for given chat`
  Chat with does not exist

SET CHATMESSAGE SEEN

Syntax
```
SET CHATMESSAGE <id> SEEN
```

Response
```
CHATMESSAGE <id> STATUS <value>
```

Parameters

- <id> – chat message ID.
- <value> – new value for chat message status; refer to CHATMESSAGE object for status values

Version
Protocol 3

Example

```
-> SET CHATMESSAGE 61 SEEN
<- CHATMESSAGE 61 STATUS READ
```

Errors

```
ERROR 18 SET: invalid WHAT
CHATMESSAGE command is missing or misspelled
```

```
ERROR 31 Unknown message id
Unknown chat message ID
```

```
ERROR 30 Invalid message id
Chat message ID is misspelled or contains non-permitted symbols (numeric are permitted)
```

```
ERROR 32 Invalid WHAT
Invalid status given to chat message, for example the message is already marked as seen
```

SET CHATMESSAGE BODY

This command enables you to change the text of a chat message.

Syntax:
```
-> SET CHATMESSAGE <chatmessage_id> BODY <text>
```

Weather a chat message text is changeable can be determined by checking the IS_EDITABLE property of a CHATMESSAGE object.

The rules for allowing editing are:

- Everyone can change their own messages.
- Creator of public chat can edit messages from others.
- Masters can edit messages originating from others, except those from the chat creator.
- Helpers and below cannot edit messages from others.

Refer to CHAT ROLES section for the list of chat roles.

Example:

```
//---------------------------------------------------------------
// First lets send out a chat message
-> CHATMESSAGE #anappo/$a1044019f5dc8c48 Test chat message
<- CHATMESSAGE 864 STATUS SENDING
<- CHAT #anappo/$a1044019f5dc8c48 ACTIVITY_TIMESTAMP 1175093328
<- CHATMESSAGE 864 STATUS SENT
//---------------------------------------------------------------
// Then lets see if we can edit it..
-> GET CHATMESSAGE 864 IS_EDITABLE
<- CHATMESSAGE 864 IS_EDITABLE TRUE
//---------------------------------------------------------------
// Then see if we can change the message text
```

```
-> SET CHATMESSAGE 864 BODY Test message after being edited
<- CHATMESSAGE 864 BODY Test message after being edited
<- CHATMESSAGE 864 EDITED_TIMESTAMP 1175093385
<- CHATMESSAGE 864 EDITED_BY anappo
<- CHATMESSAGE 864 BODY Test message after being edited
```

Version
Protocol 7 (API version 3.0)

SET MESSAGE SEEN – obsolete

Mark message as seen by the user and remove it from the missed messages list. This command is obsolete and has been replaced by the SET CHATMESSAGE SEEN command.

Syntax
`SET MESSAGE <id> SEEN`

Response
`MESSAGE <id> STATUS value`

Properties

- `<id>` – message ID;
- `value` – (new) status value

Version
Protocol 1, deprecated in protocol 3

Example

```
-> SET MESSAGE 1578 SEEN
<- MESSAGE 1578 STATUS READ
```

Errors

- `ERROR 18 SET: invalid WHAT`
  Object name missing or misspelled.
- `ERROR 30 Invalid message id`
  ID includes other than numeric characters.
- `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.

MESSAGE – obsolete

The `MESSAGE` command is obsolete and has been replaced by the CHATMESSAGE command.

Syntax
`MESSAGE <target> <text>`

Response
`CHATMESSAGE <id> STATUS SENDING` (protocol 3 and up)
`MESSAGE <id> STATUS SENDING` (protocol 1 and 2)

Parameters

- `<target>` – target username to whom to send the message
- `<text>` – message body, for example Please call me

Version
Protocol 1

Errors

```
ERROR 26 Invalid user handle
The target username is missing or includes symbols which are not premitted
```

```
ERROR 43 Cannot send empty message
The message has no body.
```

Notes
When message sending fails, a LEFT-type message is received. The message's LEAVEREASON shows why it failed. See the CHATMESSAGE object for a description.

Example

```
-> MESSAGE echo123 Please call me
<- MESSAGE 982 STATUS SENDING
<- MESSAGE 982 STATUS SENT
```

GET CHAT MEMBEROBJECTS

This command provides list of CHATMEMBER object IDs that represent chat participants.

Syntax:
```
GET CHAT <id> MEMBEROBJECTS
```

Refer to

- CHATMEMBER object for a list of CHATMEMBER properties.
- GET CHATMEMBER command for how to access CHATMEMBER properties.
- SEARCH CHATS for how to get a list of CHAT IDs.

Example:

```
-> GET CHAT #test/$test3;5f7cdbdd32dc731c MEMBEROBJECTS
<- CHAT #test/$3;5f7cdbdd32dc731c MEMBEROBJECTS 453, 454, 1465
```

Version
Protocol 7 (API version 3.0)

GET CHATMEMBER

This command provides read access to objects representing chat participants.

Syntax:
```
GET CHATMEMBER <id> <property>
```

Refer to

- CHATMEMBER object for a list of object properties.
- GET CHAT MEMBEROBJECTS command for a list of object IDs.

Example:

```
-> GET CHAT #test/$test3;5f7cdbdd32dc731c MEMBEROBJECTS
<- CHAT #test/$3;5f7cdbdd32dc731c MEMBEROBJECTS 453, 454, 1465
-> GET CHATMEMBER 1465 IDENTITY
<- CHATMEMBER 1465 IDENTITY test_p
-> GET CHATMEMBER 1465 CHATNAME
<- CHATMEMBER 1465 CHATNAME #test/$test3;5f7cdbdd32dc731c
-> GET CHATMEMBER 1465 ROLE
<- CHATMEMBER 1465 ROLE USER
-> GET CHATMEMBER 1465 IS_ACTIVE
<- CHATMEMBER 1465 IS_ACTIVE TRUE
```

Version
Protocol 7 (API version 3.0)

ALTER CHAT JOIN

This command enables you to re-join a Public chat that you have previously left. This command assumes a CHAT object is already present in the local system.

Note that this command does work with non-public multichats.
*

Syntax:
`ALTER CHAT <chat_id> JOIN`

Example:

```
//-------------------------------------------------------------------------
// Leaving public chat #anappo/$a1044019f5dc8c48
-> ALTER CHAT #anappo/$a1044019f5dc8c48 LEAVE
<- ALTER CHAT LEAVE
<- MESSAGE 392 STATUS SENDING
<- CHAT #anappo/$a1044019f5dc8c48 MEMBERS anappo2 anappo3
<- CHAT #anappo/$a1044019f5dc8c48 ACTIVEMEMBERS anappo2 anappo3
<- CHAT #anappo/$a1044019f5dc8c48 MYSTATUS UNSUBSCRIBED
<- CHAT #anappo/$a1044019f5dc8c48 STATUS UNSUBSCRIBED
<- CHAT #anappo/$a1044019f5dc8c48 BOOKMARKED FALSE
<- MESSAGE 392 STATUS SENT
//-------------------------------------------------------------------------
// Re-joining the chat
-> ALTER CHAT #anappo/$a1044019f5dc8c48 JOIN
<- CHAT #anappo/$a1044019f5dc8c48 MYSTATUS CONNECTING
<- CHAT #anappo/$a1044019f5dc8c48 STATUS UNSUBSCRIBED
<- ALTER CHAT JOIN
<- CHAT #anappo/$a1044019f5dc8c48 MEMBERS anappo2 anappo3
<- CHAT #anappo/$a1044019f5dc8c48 ACTIVEMEMBERS anappo2 anappo3
<- CHAT #anappo/$a1044019f5dc8c48 BOOKMARKED TRUE
<- CHAT #anappo/$a1044019f5dc8c48 MEMBERS anappo2 anappo3
<- CHAT #anappo/$a1044019f5dc8c48 ACTIVEMEMBERS anappo2 anappo3
<- CHAT #anappo/$a1044019f5dc8c48 MYSTATUS WAITING_REMOTE_ACCEPT
<- CHAT #anappo/$a1044019f5dc8c48 STATUS UNSUBSCRIBED
<- CHATMEMBER 75 IS_ACTIVE FALSE
<- CHATMEMBER 396 IS_ACTIVE FALSE
<- CHAT #anappo/$a1044019f5dc8c48 MEMBERS anappo anappo2 anappo3
<- CHAT #anappo/$a1044019f5dc8c48 ACTIVEMEMBERS anappo anappo3
<- CHAT #anappo/$a1044019f5dc8c48 MYSTATUS SUBSCRIBED
<- CHAT #anappo/$a1044019f5dc8c48 STATUS MULTI_SUBSCRIBED
<- CHATMEMBER 75 IS_ACTIVE TRUE
<- CHATMEMBER 396 IS_ACTIVE TRUE
<- CHAT #anappo/$a1044019f5dc8c48 ACTIVEMEMBERS anappo anappo2 anappo3
<- MESSAGE 398 STATUS READ
```

Errors

- `ERROR 504 CHAT: action failed`
  Attempt to re-join dialog or multichat. This command only enables you to re-join public chats.

Version
Protocol 7 (API version 3.0)

ALTER CHAT CLEARRECENTMESSAGES

This command clears recent chat messages in a given chat. Note that this command does not actually update user interface when a Skype client chat window for that chat is open. To see the effect, close the chat window and re-open it.

Syntax:
`ALTER CHAT <chat_id> CLEARRECENTMESSAGES`

Example:

```
-> ALTER CHAT #anappo/$test_p;297fcefb07ffc4b2 CLEARRECENTMESSAGES
<- ALTER CHAT CLEARRECENTMESSAGES
```

Version
Protocol 7 (API version 3.0)

ALTER CHAT SETALERTSTRING

This command enables you to set up a chat alert string. Normally, a small notification window will pop up at system tray when someone posts a message in a chat while the chat window is closed. When an alert string is set, the notification window will only appear when the message contains value set in `SETALERTSTRING` property.

Note that when setting this value from API, first symbol of the alert string is assumed to be "=" and gets stripped. To prevent first symbol of your alert string from being stripped, add "=" in front of it.

Syntax:
`ALTER CHAT <chat_id> SETALERTSTRING <alert_string>`

Example:

```
-> ALTER CHAT #anappo/$a1044019f5dc8c48 SETALERTSTRING "=test"
<- ALTER CHAT SETALERTSTRING
```

Version
Protocol 7 (API version 3.0)

ALTER CHAT ACCEPTADD

This command is used for accepting invitations to shared contact groups. In other chat contexts, invitations are either accepted or declined automatically, depending on user's privacy settings.

Syntax:
`ALTER CHAT <chat_id> ACCEPTADD`

Version
Protocol 7 (API version 3.0)

ALTER CHAT DISBAND

This command removes all chat participants from the chat and closes it.

Syntax:
`ALTER CHAT <chat_id> DISBAND`

Example:

```
-> ALTER CHAT #anappo/$a1044019f5dc8c48 DISBAND
<- ALTER CHAT DISBAND
<- CHAT #anappo/$a1044019f5dc8c48 MYSTATUS CHAT_DISBANDED
<- CHAT #anappo/$a1044019f5dc8c48 STATUS UNSUBSCRIBED
```

Version
Protocol 7 (API version 3.0)

Public Chats

Public Chats were introduced in API version 3.0 Public Chats are an extension of existing

multichat functionality.

From API point of view, public chats differ from multichats in that:

- CREATE command works somewhat differently, as a public chat identifier is formed differently from multichats;

Public chats have a user hierarchy with different privilege levels and a set of tools for chat administration (similar to administration of IRC channels). These administration tools are actually available for standard multichats as well (API commands such as KICK work in multichats, altho the Skype user interface for setting privileges is not available for multichats).

More or less everything listed under Creating chats and sending messages section is also applicable to public chats. The list of sections below is specific to public chats.

CHAT ROLES and PRIVILEGES

- CREATOR – member who created the chat. There can be only one creator per chat. Only creator can promote other members to masters.
- MASTER – Also known as chat hosts. Masters cannot promote other people to masters.
- HELPER – a semi-privileged member. Helpers will not be affected by the USERS_ARE_LISTENERS option. Helpers cannot promote or demote other members.
- USER – regular members who can post messages into the chat.
- LISTENER – a demoted member who can only receive messages but not post anything into the chat.
- APPLICANT – a member waiting for acceptance into the chat. Member cannot be demoted to applicants once they have been accepted.

Refer to

- ALTER CHATMEMBER CANSETROLETO command for how to determine if it is possible to change the role of any given chat member.
- ALTER CHATMEMBER SETROLETO command for more info on how to change chat member roles.

ALTER CHAT SETPASSWORD

This command enables you to set password protection to a chat channel.

Syntax:
```
ALTER CHAT <chat_id> SETPASSWORD <password> <password_hint>
```

Example:

```
-> ALTER CHAT #anappo/$a1044019f5dc8c48 SETPASSWORD test2 password is test2
<- ALTER CHAT SETPASSWORD
<- CHAT #anappo/$a1044019f5dc8c48 PASSWORDHINT password is test2
```

Note that the password must be one word – without any whitespaces in it. All subsequent words in command parameters will be considered as password hint. Password hint will be displayed to users when they join the chat.

Version
Protocol 7 (API version 3.0)

ALTER CHAT ENTERPASSWORD

This command enables you to enter passwords from within your own code, when joining password-protected chat channels.

Syntax:
```
ALTER CHAT <chat_id> ENTERPASSWORD <password>
```

Example:

```
//-------------------------------------------------------------------------
// While trying to connect to a public password-protected channel,
// we get following messages:
<- CHAT #test_l/$4ea116d4c216baef PASSWORDHINT "password is test"
<- CHAT #test_l/$4ea116d4c216baef MYSTATUS PASSWORD_REQUIRED
<- CHAT #test_l/$4ea116d4c216baef STATUS UNSUBSCRIBED
//-------------------------------------------------------------------------
// Lets supply a wrong password first and see what happens..
-> ALTER CHAT #test_l/$4ea116d4c216baef ENTERPASSWORD test2
<- ALTER CHAT ENTERPASSWORD
<- CHAT #test_l/$4ea116d4c216baef MYSTATUS CONNECTING
<- CHAT #test_l/$4ea116d4c216baef STATUS UNSUBSCRIBED
<- CHAT #test_l/$4ea116d4c216baef MYSTATUS PASSWORD_REQUIRED
<- CHAT #test_l/$4ea116d4c216baef STATUS UNSUBSCRIBED
//-------------------------------------------------------------------------
// Now lets supply correct password:
-> ALTER CHAT #test_l/$4ea116d4c216baef ENTERPASSWORD test
<- ALTER CHAT ENTERPASSWORD
<- CHAT #test_l/$4ea116d4c216baef MYSTATUS CONNECTING
<- CHAT #test_l/$4ea116d4c216baef STATUS UNSUBSCRIBED
<- CHAT #test_l/$4ea116d4c216baef MYSTATUS WAITING_REMOTE_ACCEPT
<- CHAT #test_l/$4ea116d4c216baef STATUS UNSUBSCRIBED
<- CHAT #test_l/$4ea116d4c216baef MYROLE USER
<- CHAT #test_l/$4ea116d4c216baef MEMBERS anappo test_l
<- CHAT #test_l/$4ea116d4c216baef ACTIVEMEMBERS anappo test_l
<- CHAT #test_l/$4ea116d4c216baef TIMESTAMP 1174906897
<- CHAT #test_l/$4ea116d4c216baef ADDER test_l
<- CHAT #test_l/$4ea116d4c216baef GUIDELINES test guidelines
<- MESSAGE 557 STATUS RECEIVED
<- CHAT #test_l/$4ea116d4c216baef TOPIC TestingPublicChats2
<- CHAT #test_l/$4ea116d4c216baef OPTIONS 1
<- CHATMEMBER 556 ROLE LISTENER
<- CHAT #test_l/$4ea116d4c216baef MYROLE LISTENER
<- CHATMEMBER 547 ROLE CREATOR
<- CHAT #test_l/$4ea116d4c216baef MYSTATUS SUBSCRIBED
<- CHAT #test_l/$4ea116d4c216baef STATUS MULTI_SUBSCRIBED
<- CHAT #test_l/$4ea116d4c216baef FRIENDLYNAME TestingPublicChats2
<- MESSAGE 558 STATUS RECEIVED
```

Version
Protocol 7 (API version 3.0)

ALTER CHAT SETOPTIONS

This command enables you to change chat options.

Syntax:
`ALTER CHAT <chat_id> SETOPTIONS <options bitmap>`

Chat options bits:

- 1 – JOINING_ENABLED – when this bit is off, new users cannot join the chat.
- 2 – JOINERS_BECOME_APPLICANTS – when this bit is on, new users will be able to join the chat but they will be unable to post or receive messages until authorized by one of the chat administrators (CREATOR or MASTER).
- 4 – JOINERS_BECOME_LISTENERS – when this bit is on, new users will be able to receive message in chat but unable to post until promoted to USER role. Basically a read-only flag for new users.
- 8 – HISTORY_DISCLOSED – when this bit is off, newly joined members can see chat history prior to their joining. Maximum amount of history backlog available is either 400 messages or 2 weeks of time, depending on which limit is reached first.
- 16 – USERS_ARE_LISTENERS – read-only flag for chat members with USER role.
- 32 – TOPIC_AND_PIC_LOCKED_FOR_USERS – when this bit of options is off, USER level chat members can change chat topic and the topic picture.

Example:

```
//-------------------------------------------------------------------------
// Setting flags: JOINING_ENABLED, JOINERS_BECOME_LISTENERS, HISTORY_DISCLOSED
// Adding up the bits: 1 + 4 + 8 = 13
-> ALTER CHAT #anappo/$a1044019f5dc8c48 SETOPTIONS 13
```

```
<- MESSAGE 678 STATUS SENDING
<- ALTER CHAT SETOPTIONS
<- CHAT #anappo/$a1044019f5dc8c48 OPTIONS 13
<- MESSAGE 678 STATUS SENT
```

Version
Protocol 7 (API version 3.0)

ALTER CHATMEMBER SETROLETO

This command enables chat administrators (chat CREATORS AND MASTERS) to set privilege levels (roles) for other chat members.

Syntax:

```
-> ALTER CHATMEMBER <id> SETROLETO
CREATOR|MASTER|HELPER|USER|LISTENER
```

Refer to

- Chat roles section for more information on different roles. Note that you cannot demote a user to LISTENER role when the chat is already in ready-only mode (USERS_ARE_LISTENERS chat option).
- ALTER CHATMEMBER CANSETROLETO command for how to determine if it is possible to change the role of any given chat member.

Example:

```
-> GET CHAT #anappo/$anappo3;5f7cdbdd32dc731c MEMBEROBJECTS
<- CHAT #anappo/$anappo3;5f7cdbdd32dc731c MEMBEROBJECTS 1846, 2227, 2495
-> GET CHATMEMBER 2495 IDENTITY
<- CHATMEMBER 2495 IDENTITY anappo2
-> GET CHATMEMBER 2495 ROLE
<- CHATMEMBER 2495 ROLE HELPER
-> ALTER CHATMEMBER 2495 SETROLETO USER
<- ALTER CHATMEMBER SETROLETO
<- MESSAGE 2620 STATUS SENDING
<- CHATMEMBER 2495 ROLE USER
```

Version
Protocol 7 (API version 3.0)

ALTER CHATMEMBER CANSETROLETO

This command can be used to determine weather current user is able to change the privilege level of another chat member.

Syntax:

```
-> ALTER CHATMEMBER <id> CANSETROLETO
CREATOR|MASTER|HELPER|USER|LISTENER|APPLICANT

<- ALTER CHATMEMBER CANSETROLETO TRUE|FALSE
```

Note that unlike other ALTER commands, this one doesn't actually change object properties.

Refer to

- Chat roles section for more information on different roles.
- ALTER CHATMEMBER SETROLETO command for more info on how to change chat member roles.

Example:

```
-> GET CHAT #test/$test3;5f7cdbdd32dc731c MEMBEROBJECTS
<- CHAT #test/$test3;5f7cdbdd32dc731c MEMBEROBJECTS 1846, 2227, 2495
-> GET CHATMEMBER 2495 IDENTITY
<- CHATMEMBER 2495 IDENTITY testuser
-> ALTER CHATMEMBER 2495 CANSETROLETO HELPER
```

```
<- ALTER CHATMEMBER CANSETROLETO TRUE
-> ALTER CHATMEMBER 2495 SETROLETO HELPER
<- ALTER CHATMEMBER SETROLETO
<- MESSAGE 3166 STATUS SENDING
<- CHATMEMBER 2495 ROLE HELPER
```

Version
Protocol 7 (API version 3.0)

ALTER CHAT KICK

With this command, chat member with sufficient privilege level (master or creator) can remove another member from chat.

Note that after being kicked from the channel, the kicked member can re-join the chat. For more permanent removal, see ALTER CHAT KICKBAN command.

Syntax:
`ALTER CHAT <chat_id> KICK <skypename1[, skypename2 ..]>`

Example:

```
-> ALTER CHAT  #test/$a1044019f5dc8c48 KICK test2
<- ALTER CHAT KICK
```

Version
Protocol 7 (API version 3.0)

ALTER CHAT KICKBAN

With this command, chat member with sufficient privilege level (master or creator) can permanently remove another member from chat. Note that kickban only prevents the user from re-joining the chat. Banned users can be added back to the chat by administrators from within the chat.

Syntax:
`ALTER CHAT <chat_id> KICKBAN <skypename1[, skypename2 ..]>`

Example:

```
-> ALTER CHAT  #test/$a1044019f5dc8c48 KICKBAN test2
<- ALTER CHAT KICKBAN
```

Version
Protocol 7 (API version 3.0)

ALTER CHAT FINDUSINGBLOB

This command searches for existing CHAT object with given BLOB property value and returns chat ID and status. Refer to CHAT object for more information.

Syntax:
`CHAT FINDUSINGBLOB <blob>`

Example:

```
-> CHAT FINDUSINGBLOB LsgqqqCTpxWYjt9PL1hSvGDOiPhqUuQAHxI7w7Qu7gJ3VZv_q_99ZJO4lF9Dfaw
<- CHAT #anappo2/$d936403094338dbb STATUS MULTI_SUBSCRIBED
```

Version
Protocol 7 (API version 3.0)

ALTER CHAT CREATEUSINGBLOB

This command creates a chat object, based on public chat blob. This enables you to join public chats from within your own code, assuming that you have somehow obtained the chat blob.

Syntax:

```
CHAT CREATEUSINGBLOB <blob>
```

Example:

```
//-------------------------------------------------------------------------------
// What we start is a blob of a public chat we parsed out of a
// public chat URL or, for example, got sent via another chat.
// that blob is: 6aM81Z5mZRyricRDcjkdy5bf3Y6TsCbVvaxNVVCcYSVsQxRGhlAVmTgpYexh
// First we create a CHAT object.
-> CHAT CREATEUSINGBLOB 6aM81Z5mZRyricRDcjkdy5bf3Y6TsCbVvaxNVVCcYSVsQxRGhlAVmTgpYexh
<- CHAT #anappo/$b9275b3b334341f2 NAME #anappo/$b9275b3b334341f2
<- CHAT #anappo/$b9275b3b334341f2 ACTIVITY_TIMESTAMP 0
<- CHAT #anappo/$b9275b3b334341f2 STATUS UNSUBSCRIBED
<- CHAT #anappo/$b9275b3b334341f2 TYPE MULTICHAT
<- CHAT #anappo/$b9275b3b334341f2 MYSTATUS UNSUBSCRIBED
<- CHAT #anappo/$b9275b3b334341f2 STATUS UNSUBSCRIBED
//-------------------------------------------------------------------------------
// Now that we have chat object and it's ID, we can join the chat
-> ALTER CHAT #anappo/$b9275b3b334341f2 JOIN
<- CHAT #anappo/$b9275b3b334341f2 MYSTATUS CONNECTING
<- CHAT #anappo/$b9275b3b334341f2 STATUS UNSUBSCRIBED
<- ALTER CHAT JOIN
//-------------------------------------------------------------------------------
// Note that this is our privilege level (role) in this chat
<- CHATMEMBER 293 ROLE USER
<- CHAT #anappo/$b9275b3b334341f2 MEMBERS anappo
<- CHAT #anappo/$b9275b3b334341f2 FRIENDLYNAME Avo Nappo
<- CHAT #anappo/$b9275b3b334341f2 ACTIVEMEMBERS anappo
<- CHAT #anappo/$b9275b3b334341f2 ACTIVITY_TIMESTAMP 1175004600
<- CHAT #anappo/$b9275b3b334341f2 BOOKMARKED TRUE
<- CHAT #anappo/$b9275b3b334341f2 MEMBERS anappo anappo4
<- CHAT #anappo/$b9275b3b334341f2 FRIENDLYNAME Avo Nappo, anappo4
<- CHAT #anappo/$b9275b3b334341f2 ACTIVEMEMBERS anappo anappo4
<- CHAT #anappo/$b9275b3b334341f2 MYSTATUS WAITING_REMOTE_ACCEPT
<- CHAT #anappo/$b9275b3b334341f2 STATUS UNSUBSCRIBED
<- CHATMEMBER 294 IS_ACTIVE FALSE
<- CHAT #anappo/$b9275b3b334341f2 MYROLE USER
<- CHAT #anappo/$b9275b3b334341f2 MEMBERS anappo anappo4 test_p
<- MESSAGE 298 STATUS RECEIVED
<- CHAT #anappo/$b9275b3b334341f2 ACTIVEMEMBERS anappo test_p
<- CHAT #anappo/$b9275b3b334341f2 TIMESTAMP 1175003077
<- CHAT #anappo/$b9275b3b334341f2 ADDER anappo
<- CHAT #anappo/$b9275b3b334341f2 TOPIC TestingPublicChat3
<- CHAT #anappo/$b9275b3b334341f2 OPTIONS 1
//-------------------------------------------------------------------------------
// Following notification tells us chatmember ID of the chat owner (creator)
<- CHATMEMBER 293 ROLE CREATOR
<- CHAT #anappo/$b9275b3b334341f2 MYSTATUS SUBSCRIBED
<- CHAT #anappo/$b9275b3b334341f2 STATUS MULTI_SUBSCRIBED
<- CHAT #anappo/$b9275b3b334341f2 FRIENDLYNAME TestingPublicChat3
<- MESSAGE 299 STATUS RECEIVED
<- CHATMEMBER 294 IS_ACTIVE TRUE
<- CHAT #anappo/$b9275b3b334341f2 ACTIVEMEMBERS anappo anappo4 test_p
//-------------------------------------------------------------------------------
// We can use GET CHATMEMBER 293 IDENTITY to get creator's Skypename
-> GET CHATMEMBER 293 IDENTITY
<- CHATMEMBER 293 IDENTITY anappo
//-------------------------------------------------------------------------------
// Opening chat window in UI
-> OPEN CHAT #anappo/$b9275b3b334341f2
<- OPEN CHAT #anappo/$b9275b3b334341f2
```

Version

Protocol 7 (API version 3.0)

ALTER CHAT SETGUIDELINES

This command enables you to set the Guidelines message for public chats. The guideline message is displayed at the top of the chat window.

Syntax:
```
ALTER CHAT <chat_id> SETGUIDELINES <guidelines>
```

Example:

```
-> ALTER CHAT #anappo/$a1044019f5dc8c48 SETGUIDELINES these here are test guidelines
<- MESSAGE 744 STATUS SENDING
<- ALTER CHAT SETGUIDELINES
<- CHAT #anappo/$a1044019f5dc8c48 GUIDELINES these here are test guidelines
<- MESSAGE 744 STATUS SENT
```

Version
Protocol 7 (API version 3.0)

Managing contacts and groups

Users can group contacts, for example, creating separate groups for friends, family, and work. To add a user to a group, the user must be in the contact list. Contacts can be in multiple groups at the same time. Refer to the GROUP object for a description of the object properties.

This section contains commands used for grouping the contacts.

GET GROUP USERS

The `GET GROUP USERS` command queries the members of a group.

Syntax
```
-> GET GROUP <id> USERS
```

Response
```
<- GROUP <id> USERS <user1>, <user2>, <user3>
```

Refer to SEARCH GROUPS on how to get the group ID list.

Version
Protocol 5

GET GROUP VISIBLE

The `GET GROUP VISIBLE` command queries if a group is visible to the user.

Syntax
```
-> GET GROUP <id> VISIBLE
```

Response
```
<- GROUP <id> VISIBLE {True|False}
```

Refer to SEARCH GROUPS on how to get the group ID list.

Version
Protocol 5

GET GROUP EXPANDED

The `GET GROUP EXPANDED` command queries whether a group is expanded in the Skype window.

Syntax
```
-> GET GROUP <id> EXPANDED
```

Response
```
<- GROUP <id> EXPANDED {True|False}
```

Refer to SEARCH GROUPS on how to get the group ID list.

Version
Protocol 5

GET GROUP DISPLAYNAME

The `GET GROUP DISPLAYNAME` gets the displayname for a group.

Syntax
```
-> GET GROUP <id> DISPLAYNAME
```

Response
```
<- GROUP <id> DISPLAYNAME <name>
```

Refer to SEARCH GROUPS on how to get the group ID list.

Version
Protocol 5

SET GROUP DISPLAYNAME

The `SET GROUP DISPLAYNAME` command changes the displayname for a group.

Syntax
```
-> SET GROUP <id> DISPLAYNAME <name>
```

Response
```
<- GROUP <id> DISPLAYNAME <name>
```

Refer to SEARCH GROUPS on how to get the group ID list.

Version
Protocol 5

GET GROUP TYPE

The `GET GROUP TYPE` command queries the group type.

Syntax
```
-> GET GROUP <id> TYPE
```

Response
```
<- GROUP <id> TYPE <group_type>
```

Refer to the SEARCH GROUPS command on how to get the group ID list.

Refer to the GROUP object for a list and description of group types.

Version
Protocol 5

CREATE GROUP

The `CREATE GROUP` command creates a contact group, for example a group named Family.

Syntax
```
-> CREATE GROUP <Family>
```

Response
```
<- CREATE GROUP <Family>
```

The command triggers a number of GROUP properties events:

```
<- GROUP <234> TYPE CUSTOM_GROUP
<- GROUP <234> NROFUSERS 0
<- GROUP <234> NROFUSERS_ONLINE 0
<- GROUP <234> CUSTOM_GROUP_ID <111>
<- GROUP <234> DISPLAYNAME <Family>
<- GROUP <234> USERS
```

The command triggers the following notification:
```
<- GROUP <234> USERS <user1> <user2>...
```

Version
Protocol 5

DELETE GROUP

The `DELETE GROUP` removes a contact group.

Syntax
```
-> DELETE GROUP <234>
```

Response
```
<- DELETE GROUP <234>
```

Refer to SEARCH GROUPS on how to get the group ID list.

The command triggers the following notifications:

```
<- DELETED GROUP <234>
<- GROUP <234> USERS <user1> <user2>...
```

ALTER GROUP ADDUSER

The `ALTER GROUP ADDUSER` command adds contacts to a group.

Syntax
```
-> ALTER GROUP <234> ADDUSER <userhandle|PSTN>
```

Response
```
<- ALTER GROUP <234> ADDUSER <userhandle|PSTN>
```

Parameters
```
ADDUSER <userhandle|PSTN>
```

This command triggers the following notification:
```
<- GROUP <234> NROFUSERS y
```

Note:
A contact must exist in a contactlist to be added to a group.

Refer to SEARCH GROUPS on how to get the group ID list.

Version
Protocol 5

ALTER GROUP REMOVEUSER

The `ALTER GROUP REMOVEUSER` command removes contacts from a group.

Syntax:
```
-> ALTER GROUP <group_id> REMOVEUSER <userhandle|PSTN>
<- ALTER GROUP <group_id> REMOVEUSER <userhandle|PSTN>
```

Parameters:
`REMOVEUSER <userhandle|PSTN>`

Example:

```
-> ALTER GROUP 49 REMOVEUSER anappo5
// notification - new size of group 49 is 6 contacts
<- GROUP 49 NROFUSERS 6
//  Removed user was placed in system group "Ungrouped" (group 52 in this case)
<- GROUP 52 NROFUSERS 1
<- ALTER GROUP 49 REMOVEUSER anappo5
```

Refer to SEARCH GROUPS on how to get the group ID list.

Version:
Protocol 5

Sharing contact groups

Shared contact groups differ from Send Contacts functionality in that adding users to shared groups will automatically cause cross-authorization attempts between users.

To change an existing contact group into shared contact group:

```
-> ALTER GROUP <id> SHARE [<text>]
```

Where is the contact group ID and text is invitation message displayed to the invited user.

To accept invitation to a shared group:

```
-> ALTER GROUP <id> ACCEPT
```

To decline invitation to a shared group:

```
-> ALTER GROUP <id> DECLINE
```

Refer to SEARCH GROUPS on how to get the list of group IDs.

Version
Protocol 6

SET USER DISPLAYNAME

The `SET USER DISPLAYNAME` command changes the display name of a contact.

By default this USER object property is empty. If a value is assigned to this property with SET <skypename> DISPLAYNAME <value> then that value will be displayed in Skype UI instead of user's FULLNAME.

Syntax:
```
-> SET USER <handle|PSTN> DISPLAYNAME <name>

<- SET USER <handle|PSTN> DISPLAYNAME <name>
```

Version
Protocol 5

Search commands

The search command requests specific information about objects. If no target is specified, all results for specified objects are returned.

Syntax:
```
SEARCH USERS|FRIENDS|CALLS [<target>]|ACTIVECALLS|MISSEDCALLS|
VOICEMAILS|CHATS|MISSEDCHATS|ACTIVECHATS|RECENTCHATS|
BOOKMARKEDCHATS|CHATMESSAGES [<target>]|MISSEDCHATMESSAGES|MESSAGES
[<target>]|MISSEDMESSAGES|USERSWAITINGMYAUTHORIZATION|GROUPS [{ ALL
| CUSTOM | HARDWIRED }]|FILETRANSFERS|ACTIVEFILETRANSFERS|SMSS|
MISSEDSMSS
```

Notes

- In Skype for Windows 1.1 only one search at a time is allowed; since version 1.2 multiple searches can executed at the same time;
- The number of search results is not limited.
- SkypeOut contacts: since Skype for Windows 1.2 release it is possible to get the list of SkypeOut contacts which are part of the main contact list and they are returned with the contact list numbers, if the SEARCH FRIENDS command is executed. To get more information about the number in a current user's SkypeOut contacts use the GET USER <number> <fullname>" command.
  This section contains the search commands.

SEARCH FRIENDS

Syntax
```
SEARCH FRIENDS
```

Response
```
USERS [user[, user]*]
```
returns a list of found usernames; an empty list if no match is found

- ERROR 67 target not allowed with SEARCH FRIENDS
  A target(such as mike) was specified with the SEARCH FRIENDS command

Version
Protocol 1

Example

```
-> SEARCH FRIENDS
<- USERS tim, joe, mike
```

SEARCH USERS

Syntax
```
-> SEARCH USERS <target>
```

Parameters
<target> – part of username or e-mail to match. If the search string contains "@", the search is performed by e-mail address and has to be an exact match. If the search string is a valid Skype username, the search is performed on the full name and username fields. In all other cases the search is made on the full name field only.

Response
<- USERS [<username>[, <username>]*]
returns a list of found usernames; list is empty if no match was found

Errors

- ERROR 4 Empty target not allowed
  Target username is not specified

Notes
When running the SEARCH USERS command, USER notifications are reported back to the API client as users are found on the network. The API client should ignore these events and request each user's property after the search.

Version
Protocol 1

Example:

```
-> SEARCH USERS echo123
<- USERS echo123, echo1232885
```

SEARCH CALLS

Syntax
SEARCH CALLS <target>

Parameters
<target> – username. Specifying a target is optional. If a target is specified, Skype searches the call history between the current user and the target user.

Response
<- CALLS [id[, id]*]
Returns a list of call IDs. If a target is specified, Skype returns IDs of all calls that have been made between the current and target user.

Errors

- ERROR 5 Search CALLS: invalid target
  Characters that are not permitted were used in the target username. The username must have 6-22 characters and can contain only the following symbols: {a-Z, 0-9-_,}.

Version
Protocol 1

Example

```
-> SEARCH CALLS abc
<- CALLS 15, 16, 39
```

SEARCH ACTIVECALLS

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

Syntax
-> SEARCH ACTIVECALLS

Response
`<- CALLS [<id>[, <id>]*]`
Returns a list of active call IDs.

Errors

- `ERROR 3 SEARCH: unknown WHAT`
  `ACTIVECALLS` was misspelled.

Version
Protocol 1

Example

```
-> SEARCH ACTIVECALLS
<- CALLS 25, 56
```

SEARCH MISSEDCALLS

Syntax
`-> SEARCH MISSEDCALLS`

Response
`<- CALLS [<id>[, <id>]*]`
Returns a list ofmissed call IDs, calls in `MISSED` status.

Errors

- `ERROR 6 SEARCH MISSEDCALLS: target not allowed`
  No target is allowed with `SEARCH MISSEDCALLS`.

Version
Protocol 1

Example

```
-> SEARCH MISSEDCALLS
<- CALLS 25, 56
```

SEARCH SMSS

All SMS messages that you have created in Skype remain stored in the system until they get removed with [#MANAGING_SMS_MESSAGES_DELETING DELETE SMS] command.

The list of these SMS messages can be queried with `SEARCH SMSS` command:

Syntax:
`-> SEARCH SMSS`

`<- SMSS <ID1>, <ID2>, <ID3> ..`

Example:

```
-> SEARCH SMSS
<- SMSS 233
```

Refer to SMS object section for a list of SMS object properties.

Version
Added in API version 2.5

SEARCH MISSEDSMSS

Returns a list of IDs of received but unread SMS objects.

Syntax:

```
-> SEARCH MISSEDSMSS

<- SMSS <ID1>, <ID2>, <ID3> ..
```

Example:

```
-> SEARCH SMSS
<- SMSS 233
```

Refer to SMS object section for a list of SMS object properties.

Version
Added in API version 2.5

SEARCH VOICEMAILS

Returns a list of voicemail IDs.

Syntax:

```
-> SEARCH VOICEMAILS

<- VOICEMAILS [<id>[, <id>]*]
```

Errors

- `ERROR 29 SEARCH VOICEMAILS: target not allowed`
  No target is allowed with `SEARCH VOICEMAILS`.

Version
Protocol 5

Example:

```
-> SEARCH VOICEMAILS
<- VOICEMAILS 65, 70, 71
```

SEARCH MISSEDVOICEMAILS

Returns a list of IDs of missed voicemails.

Syntax:

```
-> SEARCH MISSEDVOICEMAILS

<- VOICEMAILS [<id>[, <id>]*]
```

Errors

- `ERROR 29 SEARCH MISSEDVOICEMAILS: target not allowed`
  No target is allowed with `SEARCH MISSEDVOICEMAILS`

Version
Protocol 6

Example:

```
-> SEARCH MISSEDVOICEMAILS
<- VOICEMAILS 65, 70, 71
```

SEARCH MESSAGES

Syntax
`SEARCH MESSAGES [<target>]`

Parameters

- `<target>` – username. It is optional to specify a target. If a target is specified, Skype searches the message history between the current user and the target user.

Response
`MESSAGES [<id>[, <id>]*]`
Returns a list of message IDs. If a target is specified, Skype returns IDs of all messages that have been sent between the current user and the target user.

Errors

- `ERROR 5 SEARCH MESSAGES: invalid target`
  A character was used in the target username that is not permitted. The username must have 6-22 characters and can contain only the following symbols: {a-Z, 0-9-_.}.

Version
Protocol 1, deprecated in protocol 3

Notes
This search is deprecated in protocol 3, use the SEARCH CHATMESSAGES command instead.

Example

```
-> SEARCH MESSAGES abc
<- MESSAGES 123, 124
```

SEARCH MISSEDMESSAGES

Syntax
`SEARCH MISSEDMESSAGES`

Response
`MESSAGES [<id>[, <id>]*]`
Returns a list of message IDs.

Errors

- `ERROR 29 SEARCH MISSEDMESSAGES: target not allowed`
  No target is allowed with the `SEARCH MISSEDMESSAGES` command.

Version
Protocol 1, deprecated in protocol 3

Notes
This search is deprecated in protocol 3. Use the `SEARCH MISSEDCHATMESSAGES` command instead.

Example

```
-> SEARCH MISSEDMESSAGES
<- MESSAGES 123, 124
```

SEARCH CHATS

Syntax
`SEARCH CHATS`

Response
`CHATS [<chatname>[, <chatname>]*]`
Returns a list of chat IDs.

Errors

- `ERROR 107 target not allowed with CHATS`
  No target isallowed with the `SEARCH CHATS` command.

Version
Protocol 3

Example

```
-> SEARCH CHATS
<- CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391
```

SEARCH ACTIVECHATS

Syntax
`SEARCH ACTIVECHATS`

Response
`CHATS [<chatname>[, <chatname>]*]`
Returns a list of chat IDs that are open in the window.

Errors

- `ERROR 29 No target allowed`
  No target is allowed with `SEARCH ACTIVECHATS` .

Version
Protocol 5

Example

```
-> SEARCH ACTIVECHATS
<- CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391
```

SEARCH MISSEDCHATS

Syntax
`SEARCH MISSEDCHATS`

Response
`CHATS [<chatname>[, <chatname>]*]`
Returns a list of chat IDs that include unread messages.

Errors

- `ERROR 29 SEARCH MISSEDCHATS: target not allowed`
  Notarget is allowed with `SEARCH MISSEDCHATS` .

Version
Protocol 5

Example

```
-> SEARCH MISSEDCHATS
<- CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391
```

SEARCH RECENTCHATS

Syntax
`SEARCH RECENTCHATS`

Response
`CHATS [<chatname>[, <chatname>]*]`
Returns a list of recent chat IDs.

Errors

- `ERROR 29 SEARCH RECENTCHATS: target not allowed`
  Notarget is allowed with `SEARCH RECENTCHATS` .

Version
Protocol 5

Example

```
-> SEARCH RECENTCHATS
<- CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391
```

SEARCH BOOKMARKEDCHATS

Syntax
`SEARCH BOOKMARKEDCHATS`

Response
`CHATS [<chatname>[, <chatname>]*]`
Returns a list of bookmarked chat IDs.

Errors

- `ERROR 29 SEARCH BOOKMARKEDCHATS: target not allowed`
  Notarget is allowed with `SEARCH BOOKMARKEDCHATS` .

Version
Protocol 5

Example

```
-> SEARCH BOOKMARKEDCHATS
<- CHATS #bitman/$jessy;eb06e65612353279, #bitman/$jdenton;9244e98f82d7d391
```

SEARCH CHATMESSAGES

Syntax
`SEARCH CHATMESSAGES [<username>]`

Parameters
`<username>` – target username, optional. If a username is specified, only chatmessages from/to that target user are returned.

Response
`CHATMESSAGES [<id>[, <id>]*]`
Returns a list of chat message IDs.

Errors

- `ERROR 29 SEARCH CHATMESSAGES: Target not allowed`
  The target username contained a character that is not permitted. (Username must have 6-22 characters and can contain only the following symbols: {a-Z, 0-9-_,.}.

Version
Protocol 3

Example

```
-> SEARCH CHATMESSAGES abc
<- CHATMESSAGES 60, 59
```

SEARCH MISSEDCHATMESSAGES

Syntax
`SEARCH MISSEDCHATMESSAGES`

Response
`CHATMESSAGES [<id>[, <id>]*]`
Returns a list of missed chat message IDs.

Errors

- `ERROR 29 SEARCH MISSEDCHATMESSAGES: target not allowed`
  Notarget is allowed with `SEARCH MISSEDCHATMESSAGES`.

Version
Protocol 3

Example

```
-> SEARCH MISSEDCHATMESSAGES
<- CHATMESSAGES 61, 62
```

SEARCH USERSWAITINGMYAUTHORIZATION

List of users who are waiting for contact authorization.

Syntax:

```
-> SEARCH USERSWAITINGMYAUTHORIZATION

<- USERS [<skypename1>[, <skypename2>]*]
```

Errors

- `ERROR 29 SEARCH USERSWAITINGMYAUTHORIZATION: target not allowed`

Version
Protocol 5

Example:

```
-> SEARCH USERSWAITINGMYAUTHORIZATION
<- USERS tim, john, echo123
```

SEARCH GROUPS

The `SEARCH GROUPS` command returns comma-separated list of IDs of user's contact groups.

Syntax:

```
-> SEARCH GROUPS [{ ALL | CUSTOM | HARDWIRED }]

<- GROUPS <id1>, <id2>, <id3>, ...
```

Example:

```
//-----------------------------------------------------
// Getting a list of custom (user-made) groups
```

```
-> SEARCH GROUPS CUSTOM
<- GROUPS 3238, 3239, 3240, 3241, 3242, 3372
//-----------------------------------------------------
// Getting group names from IDs goes like this:
-> GET GROUP 3240 DISPLAYNAME
<- GROUP 3240 DISPLAYNAME test
```

Version
Protocol 5

Errors

```
ERROR 561 - SEARCH GROUPS: invalid target
ERROR 562 - Invalid group id
ERROR 563 - Invalid group object
ERROR 564 - Invalid group property given
```

SEARCH FILETRANSFERS

Returns a list of all file transfer IDs. Refer to FILETRANSFER object for more details.

Syntax
```
-> SEARCH FILETRANSFERS
```

Response
```
<- FILETRANSFERS [<id>[, <id>]*]
```

Example:

```
-> SEARCH FILETRANSFERS
<- FILETRANSFERS 1343, 1314, 1263, 1249, 1241, 982, 544, 1086
```

Version
Protocol 7 (API version 3.0)

SEARCH ACTIVEFILETRANSFERS

Returns a list of currently active (ones that are nor `COMPLETED`, `CANCELLED` or `FAILED`) file transfer IDs.
Refer to FILETRANSFER object for more details.

Note that it is not necessary for remote users to accept the file transfer for it to become listed in ACTIVEFILETRANSFERS for both parties.

Syntax
```
-> SEARCH ACTIVEFILETRANSFERS
```

Response
```
<- FILETRANSFERS [<id>[, <id>]*]
```

Example:

```
-> SEARCH ACTIVEFILETRANSFERS
<- FILETRANSFERS 1411
-> GET FILETRANSFER 1411 STATUS
<- FILETRANSFER 1411 STATUS WAITING_FOR_ACCEPT
```

Version
Protocol 7 (API version 3.0)

Managing history

These commands are available to clear chat, voicemail, and call history.

Skype4Com example:

- CallHistory.pas

CLEAR CHATHISTORY

This command clears chat history. **NB!** This command does not remove chat entries from the Skype history tab. Instead, it clears chatmessage histories within chats.

Syntax

```
-> CLEAR CHATHISTORY

<- CLEAR CHATHISTORY
```

CLEAR VOICEMAILHISTORY

Clears voicemail entries from the history tab in Skype UI.

Syntax:

```
-> CLEAR VOICEMAILHISTORY

<- CLEAR VOICEMAILHISTORY
```

Example:

```
-> CLEAR VOICEMAILHISTORY
<- CLEAR VOICEMAILHISTORY
<- VOICEMAIL 3398 STATUS DELETING
```

CLEAR CALLHISTORY

Clears call entries from the history tab in Skype UI.

Syntax:

```
-> CLEAR CALLHISTORY <ALL|MISSED|INCOMING|OUTGOING> [skypename]

<- CLEAR CALLHISTORY <ALL|MISSED|INCOMING|OUTGOING> [skypename]
```

Example:

```
//------------------------------------------------------------
// Removes incoming calls from user test2 from Skype history tab
-> CLEAR CALLHISTORY INCOMING test2
<- CLEAR CALLHISTORY INCOMING test2
```

Controlling Skype user interface

This section lists the commands used to control the Skype user interface.

FOCUS

The `FOCUS` brings the Skype window into focus on screen (on top).

Syntax:

```
-> FOCUS
```

```
<- FOCUS
```

See also SET WINDOWSTATE command for more recent and universal version of the same functionality.

Note also that from version 3.6 the FOCUS command produces additional window state notification message in following format:

```
<- WINDOWSTATE NORMAL|MINIMIZED|MAXIMIZED|HIDDEN
```

Version
Protocol 1

MINIMIZE

This command minimizes the main Skype window into the system tray.

Syntax:

```
-> MINIMIZE

<- MINIMIZE
*
```

See also SET WINDOWSTATE command for more recent and universal version of the same functionality.

Note also that from version 3.6 the MINIMIZE command produces additional window state notification message in following format:

```
<- WINDOWSTATE NORMAL|MINIMIZED|MAXIMIZED|HIDDEN
```

Version
Skype for Windows 1.3

Notes
This command does not minimize other Skype windows, such as chat or filetransfer.

GET WINDOWSTATE

Returns the current state of the Skype main window. The WINDOWSTATE property is read-write, so you can cause the Skype main window to minimize to system tray, maximize, etc. with corresponding SET WINDOWSTATE command.

Syntax:

```
@→ GET WINDOWSTATE @

<- WINDOWSTATE NORMAL|MINIMIZED|MAXIMIZED|HIDDEN
```

Example:

```
-> #1 GET WINDOWSTATE
<- #1 WINDOWSTATE NORMAL
-> #2 SET WINDOWSTATE MINIMIZED
<- #2 WINDOWSTATE MINIMIZED
<- WINDOWSTATE MINIMIZED
-> #3 SET WINDOWSTATE MAXIMIZED
<- #3 WINDOWSTATE MAXIMIZED
<- WINDOWSTATE MAXIMIZED
<- WINDOWSTATE MINIMIZED
-> #4 SET WINDOWSTATE NORMAL
<- #4 WINDOWSTATE NORMAL
<- WINDOWSTATE NORMAL
```

Note that you also get these WINDOWSTATE notification messages when the window state was altered via UI, i.e. when a user clicks on minimize button in the Skype window, corresponding API notification event is generated.

This is also the reason SET WINDOWSTATE command receives two reply notifications. One is sent as direct reply to the actual API command, the second one is generated by the change in Skype window state.

NB! As seen in the example above, of those two notification events, in response to SET WINDOWSTATE only one comes with command identifier.

Version:

Windows version 3.6

SET WINDOWSTATE

This command causes the Skype Main window to change state. Note that this command only applies to Skype **main** window. Other Skype windows, such as chat windows or file trasfer windows are unaffected by this command.

Syntax:
@→ SET WINDOWSTATE NORMAL|MINIMIZED|MAXIMIZED|HIDDEN @

<- WINDOWSTATE NORMAL|MINIMIZED|MAXIMIZED|HIDDEN

<- WINDOWSTATE NORMAL|MINIMIZED|MAXIMIZED|HIDDEN

Note this command generates two reply notifications. If you are using this command together with command identifiers, then it might be important to know that only the first one of those notifications comes back with command ID (see example below).

Parameters:

- NORMAL – resets Skype main window to previous manually adjusted size and position.
- MINIMIZED – minimizes Skype window to taskbar – NB! This does not put Skype to system tray, to minimize Skype to system tray use HIDDEN parameter.
- MAXIMIZED – maximizes Skype main window all over the current desktop.
- HIDDEN – minimizes Skype main window to system tray.

Example:

```
-> #1 GET WINDOWSTATE
<- #1 WINDOWSTATE NORMAL
-> #2 SET WINDOWSTATE MINIMIZED
<- #2 WINDOWSTATE MINIMIZED
<- WINDOWSTATE MINIMIZED
-> #3 SET WINDOWSTATE MAXIMIZED
<- #3 WINDOWSTATE MAXIMIZED
<- WINDOWSTATE MAXIMIZED
<- WINDOWSTATE MINIMIZED
-> #4 SET WINDOWSTATE NORMAL
<- #4 WINDOWSTATE NORMAL
<- WINDOWSTATE NORMAL
```

Version:

*Added in API version 3.6

OPEN ADDAFRIEND

This command opens the Add a Contact window. **NB!** Don't miss that "A" between "ADD" and "FRIEND".

Syntax:

```
-> OPEN ADDAFRIEND [<username>]

<- OPEN ADDAFRIEND [<username>]
```

Parameters

`<username>` – target username is optional. If a username is specified, the window is prefilled with it.

Errors

`ERROR 69 OPEN: invalid WHAT`
Open target is missing or misspelled

Version

Skype for Windows 1.0

OPEN IM

This command opens the chat window with prefilled message.

Syntax:

```
-> OPEN IM <username> [<message>]
```

Response:

In response, open chat command feedback is generated, followed by with echoing back the original command (see example below).

Parameters

- `<username>` – contact username to whom to send the message.
- `<message>` – optional message body prefilled into the window. Note that this message is not actually sent – just pasted into chat window's input line.

Errors

- `ERROR 69 OPEN: invalid WHAT`
  Open target is missing or misspelled
- `ERROR 70 Invalid user handle`
  Username is missing or contains not permitted symbols

Notes

The protocol 5 chat management commands and Skype for Windows 1.3 `OPEN CHAT` command are preferable to the `OPEN IM` command.

Example:

```
-> OPEN IM echo123 this is a prefilled chatmessage
<- CHAT #anappo/$echo123;ebe5311cdd203657 NAME #anappo/$echo123;ebe5311cdd203657
<- CHAT #anappo/$echo123;ebe5311cdd203657 ACTIVITY_TIMESTAMP 0
<- MESSAGE 1259 STATUS SENDING
<- CHAT #anappo/$echo123;ebe5311cdd203657 TYPE DIALOG
<- CHATMEMBER 1257 ROLE USER
<- CHAT #anappo/$echo123;ebe5311cdd203657 MYROLE USER
<- CHAT #anappo/$echo123;ebe5311cdd203657 ACTIVEMEMBERS anappo
<- CHAT #anappo/$echo123;ebe5311cdd203657 MYSTATUS SUBSCRIBED
<- CHAT #anappo/$echo123;ebe5311cdd203657 STATUS DIALOG
<- CHAT #anappo/$echo123;ebe5311cdd203657 TIMESTAMP 1178793154
<- CHAT #anappo/$echo123;ebe5311cdd203657 DIALOG_PARTNER echo123
<- CHAT #anappo/$echo123;ebe5311cdd203657 MEMBERS anappo echo123
<- CHAT #anappo/$echo123;ebe5311cdd203657 FRIENDLYNAME Echo / Sound Test Service
<- OPEN IM echo123 this is a prefilled chatmessage
```

Version
Skype for Windows 1.0

OPEN CHAT

Opens chat window for existing CHAT object.

Syntax:
```
-> OPEN CHAT <chat_id>

<- OPEN CHAT <chat_id>
```

Parameters
`<chat_id>` – existing chat identifier (Refer to SEARCH CHATS command).

Errors

- `ERROR 69 invalid open what`
  Open target is missing or misspelled
- `ERROR 105 Invalid chat name`
  Chat id is missing or chat with this id doesn't exist.

Version
Skype for Windows 1.3

Example:

```
-> OPEN CHAT #test/$echo123;52c2750d8686c10c
<- OPEN CHAT #test/$echo123;52c2750d8686c10c
```

NB! From version 3.6 and later, opening chat windows (both from API and manually via UI) generate additional chat window open and close notfication messages. Refer to the Chat notifications section for more information.

OPEN FILETRANSFER

Syntax:
```
-> OPEN FILETRANSFER <username>[, <username>]*[ IN <folder>]

<- OPEN FILETRANSFER <username>[, <username>]*[ IN <folder>]
```

Parameters

- `<username>` – list of usernames to transfer file to;
- `<folder>` – optional, filesystem folder for file selection window. If not specified, the file transfer window opens in the default directory.

Errors

- `ERROR 69 invalid open what`
  Open target is missing or misspelled
- `ERROR 108 user not contact`
  Command is allowed for authorized contacts only
- `ERROR 109 directory doesn't exist`
  Given folder does not exist or user has no access to it

Example:

```
-> OPEN FILETRANSFER echo123 IN C:\temp
<- ERROR 108 user not contact
-> OPEN FILETRANSFER myfriend IN C:\temp
<- OPEN FILETRANSFER myfriend IN C:\temp
```

Version
Skype for Windows 1.3

OPEN LIVETAB

Opens Live tab in Skype UI.

Syntax:
```
-> OPEN LIVETAB

<- OPEN LIVETAB
```

Version
API version 3.2 (protocol 7)

OPEN VIDEOTEST

This command opens the Video test window to test if video is working. See OPEN VIDEOTEST command reference for details.

OPEN VOICEMAIL

This command brings the callhistory tab into focus and starts playing a voicemail. See OPEN VOICEMAIL command reference for details.

OPEN PROFILE

This command opens the profile window for the current user.

Syntax:
```
-> OPEN PROFILE

<- OPEN PROFILE
```

Errors

- `ERROR 69 invalid open what`
  Open target is missing or misspelled

Version
Skype for Windows 1.4

OPEN USERINFO

This command opens the profile window for a named Skype contact. Note that when the contact given in the parameter does not exist, a profile window is still opened, with an option to add `<skypename>` to user's contact list. Therefore, you cannot rely on feedback of this command to determine whether `<skypename>` is present in your contact list.

Syntax:
```
-> OPEN USERINFO <skypename>

<- OPEN USERINFO <skypename>
```

Parameters
`<skypename>` – Skypename of contact

Errors

- ERROR invalid skypename
- ERROR 69 invalid open what
- Open target is missing or misspelled

Version
Skype for Windows 1.4

OPEN CONFERENCE

This command opens the create conference window. Note that this command does not allow parameters.

Syntax:
```
-> OPEN CONFERENCE

<- OPEN CONFERENCE
```

Errors

- ERROR 69 invalid open what
- Open target is missing or misspelled

Version
Skype for Windows 1.4

OPEN SEARCH

This command opens the Skype user search window. Note that this command does not allow parameters.

Syntax:
```
-> OPEN SEARCH

<- OPEN SEARCH
```

Errors

- ERROR 69 invalid open what
- Open target is missing or misspelled

Version
Skype for Windows 1.4

OPEN OPTIONS

This command opens the options configuration window.

Syntax:
```
-> OPEN OPTIONS <page>

<- OPEN OPTIONS <page>
```

Parameters
<page>, possible values:

- general
- privacy
- notifications
- soundalerts
- sounddevices
- hotkeys
- connection

- voicemail
- callforward
- video
- advanced

Note that no error feedback is generated that when an erroneous page name is passed in the <page> parameter – the command will still be echoed back, it simply does nothing.
*

Errors

- ERROR 69 invalid open what
- Open target is missing or misspelled

Version

Skype for Windows 1.4
Note: `OPEN OPTIONS video` parameter was introduced in Skype for Windows 2.0.

OPEN CALLHISTORY

This command opens and sets the focus to the call history tab in the main Skype window.

Syntax:

```
-> OPEN CALLHISTORY

<- OPEN CALLHISTORY
```

Errors

- ERROR 69 invalid open what Open target is missing or misspelled

Version

Skype for Windows 1.4

As of version 2.0.0.12, this command also works on Linux.

OPEN CONTACTS

This command opens and sets the focus to the contacts tab in the main Skype window.

Syntax:

```
-> OPEN CONTACTS

<- OPEN CONTACTS
```

Errors

- ERROR 69 invalid open what
- Open target is missing or misspelled

Version

Skype for Windows 1.4

As of version 2.0.0.12, this command also works on Linux.

OPEN DIALPAD

This command opens and sets the focus to the dialpad tab in the main Skype window.

Syntax:

```
-> OPEN DIALPAD
```

```
<- OPEN DIALPAD
```

Errors

- ERROR 69 invalid open what
- Open target is missing or misspelled

Version
Skype for Windows 1.4

OPEN SENDCONTACTS

This command opens the send contacts window.

Syntax:
```
-> OPEN SENDCONTACTS <username> [ <username2> <username3>]

<- OPEN SENDCONTACTS <username> [ <username2> <username3>]
```

Parameters
Whitespace separated list of Skype usernames of recepients of the contact list.

Errors

- ERROR 69 invalid open what
- Open target is missing or misspelled
- ERROR 4 OPEN Empty target not allowed – missing username parameter(s)

Version
Skype for Windows 1.4

OPEN BLOCKEDUSERS

This command opens the blocked users tab of the Options window.

Syntax:
```
-> OPEN BLOCKEDUSERS

<- OPEN BLOCKEDUSERS
```

Errors

- ERROR 69 invalid open what
- Open target is missing or misspelled

Version
Skype for Windows 1.4

OPEN IMPORTCONTACTS

This command opens the import contacts wizard.

Syntax:
```
-> OPEN IMPORTCONTACTS

<- OPEN IMPORTCONTACTS
```

Errors

- ERROR 69 invalid open what
- Open target is missing or misspelled

Version
Skype for Windows 1.4

OPEN GETTINGSTARTED

This command opens the getting started wizard.

Syntax:
```
-> OPEN GETTINGSTARTED

<- OPEN GETTINGSTARTED
```

Errors

- `ERROR 69 invalid open what`
- `Open target is missing or misspelled`

Version
Skype for Windows 1.4

OPEN AUTHORIZATION

This command opens the authorization request window for a given user.

Syntax:
```
-> OPEN AUTHORIZATION <skypename>

<- OPEN AUTHORIZATION <skypename>
```

Parameters
`skypename` of the user whose authorization is requested.

Errors

- `ERROR 69 invalid open what`
- `Open target is missing or misspelled`
- `ERROR 117 OPEN User already authorized`

Version
Skype for Windows 1.4

BTN_PRESSED and BTN_RELEASED

BTN_PRESSED command does not actually do anything useful. BTN_RELEASED command can be used to simulate keyboard events in Skype UI.

Syntax:
```
-> BTN_PRESSED <key>

<- BTN_PRESSED <key>

-> BTN_RELEASED <key>

<- BTN_RELEASED <key>
```

Parameters:

Parameter can have one the of following values:
*

{0...9 | A...Z | # | * | + | UP | DOWN | YES | NO | SKYPE | PAGEUP | PAGEDOWN}

Note that during an active call, when either Call or Call Phone tabs are focused, `BTN_RELEASED` command with parameter that is a valid DTMF code, will cause that DTMF code to be sent to the remote party of the call.

Version
Protocol 5

GET CONTACTS FOCUSED

This command returns the skypename of a contact currently focused in Skype UI. Note that when more than one contacts are selected in SKype UI, this command only returns only one contact (the last one focused).

Syntax:

```
-> GET CONTACTS_FOCUSED

<- CONTACTS FOCUSED <skypename>
```

Note that the `<- CONTACTS FOCUSED` response has the same syntax as automatic focus notifications.

Version
Protocol 7 (API version 3.1)

GET/SET UI_LANGUAGE

Following two commands are available to change and retrieve current interface language settings:

```
-> GET UI_LANGUAGE

-> SET UI_LANGUAGE <iso2>
```

Example:

```
-> GET UI_LANGUAGE
<- UI_LANGUAGE en
-> SET UI_LANGUAGE en
<- UI_LANGUAGE en
<- UI_LANGUAGE en
<- UI_LANGUAGE en
```

Note that the `<- UI_LANGUAGE <iso2>` notification message is also generated by Skype when language settings get changed manually from the user interface.

NB! When the UI language is set via custom language file, `GET UI_LANGUAGE` will return “xx” (used to return “en” in versions prior to 3.5).

Version
Protocol 6 (API version 3.0)

GET/SET WALLPAPERS

Following two commands are available to change and retrieve current interface wallpapers:

```
-> GET WALLPAPER

<- WALLPAPER [<filename>]
```

```
-> SET WALLPAPER [<filename>]

<- WALLPAPER [<filename>]
```

Note that the filename parameter must contain full path as well as file extension of the wallpaper file. The filename parameter must not be enclosed in quotes.

When `SET WALLPAPER` command is given without a parameter, it will remove current wallpaper.

Supported picture formats are PNG, JPG, and BMP

Example:

```
//-------------------------------------------------------
// Setting user interface background
-> SET WALLPAPER C:\Stuff\test.bmp
<- WALLPAPER C:\Stuff\test.bmp
<- WALLPAPER C:\Stuff\test.bmp
//-------------------------------------------------------
// Trying non-existing file..
-> SET WALLPAPER c:\Stuff\wrongfile.bmp
<- ERROR 111 SET File not found
//-------------------------------------------------------
// Retrieving background filename
-> GET WALLPAPER C:\Stuff\test.bmp
<- WALLPAPER C:\Stuff\test.bmp
//-------------------------------------------------------
// Clearing background filename
-> SET WALLPAPER
<- WALLPAPER
<- WALLPAPER
```

Note that the `<- WALLPAPER <filename>` notification message is also generated by Skype when the wallpaper is changed manually from the user interface.

Version
Protocol 6 (API version 3.0)

SILENT_MODE

While in silent mode, the Skype client will no longer send out any visual notifications of calls, chat messages or other Skype events, although you will still hear ringtone when someone is calling you.

Syntax:

```
-> SET SILENT_MODE {ON|OFF}

<- SILENT_MODE {ON|OFF}
```

Example:

```
-> SET SILENT_MODE ON
<- SILENT_MODE ON
-> SET SILENT_MODE OFF
<- SILENT_MODE OFF
```

Silent mode can also be turned off by doubleclicking on the Skype icon in the System Tray. NB! Using 'Open Skype' command from the System Tray local menu will not turn Silent Mode off. Only double-click on the icon does.

Note that when a user manually turns off silent mode from System Tray, SILENT_MODE OFF notification is sent out by Skype.

Note that switching silent mode ON will cause the Skype Client to pop a confirmation

message, displaying the name of the application from which the silent mode request originated. This confirmation message will re-pop every time a third party application tries to enter silent mode.

Version
Protocol 6 (Skype API 2.6)

Custom Menus and Events

In API version 3.0, it is possible to add your own menu items under !DoMore sections of Skype UI menus. When such menu items get clicked on by a user, notification events are sent back to application from which the menu was created. A companion functionality to this are Skype Alert Events – clickable notification event entries in Skype UI that you can add and remove from your own code.

Custom Menu Items

The custom menu interface provides commands, notifications and events required to create and manage custom menu entries in the Skype client. Custom menu items are automatically removed when the API client that created them is disconnected.

When a custom menu item is clicked by the user, notification event to the API client is fired. Each API client has its own specific menu items and each client only receives notifications from menu items it creates.

The menu items can appear in Do More sections of various menus across the Skype user interface. Which particular Do More menu receives the menu item is controlled by the `CONTEXT` parameter of the CREATE MENU_ITEM command.

Note that custom menus are currently only supported by Windows client.

Version
Protocol 6 (API version 3.0)

CREATE MENU_ITEM

Creates a custom menu item in one of the Do More menus of the Skype interface.

Syntax:
```
CREATE MENU_ITEM <id> CONTEXT <context> CAPTION <caption> [HINT
<hint>]
*
[ICON <icon_path>] [ENABLED true|false] [ENABLE_MULTIPLE_CONTACTS
true|false] [CONTACT_TYPE_FILTER skype|skypeout|all]
```

Example:

```
-> CREATE MENU_ITEM test01 CONTEXT contact CAPTION "TEST 01" ENABLED true
<- MENU_ITEM test01 CREATED
//-------------------------------------------------------------
// Following menu item will only be enabled for SkypeOut contacts
-> CREATE MENU_ITEM test02 CONTEXT contact CAPTION "TEST FOR SKYPEOUT" CONTACT_TYPE_FILTER
skypeout
<- MENU_ITEM test02 CREATED
```

Parameters of the `CREATE MENU_ITEM` command:

- `ID` – Unique alphanumeric identifier, must start with a letter.
- `CONTEXT` – controls in which one of the Do More menus the menu item will appear. Valid values are:
  - `CHAT` – Do More button at the upper part of a Chat window. Note that the Do More button is disabled when there are more than two chat participants.
  - `CALL` – Do More menu at the upper part of the Call tab.

- MYSELF – Do More menu from the Personalize button on user's moodmessage tab.
- TOOLS – Do More sub-menu under the Tools menu.
- CONTACT – Do More sub menu from contact menu that can be opened by right-clicking on a contact. Note the Do More menu is disabled when more that one contacts are selected. The CONTACT key has three sub-keys:
  - SKYPE – menu item will be enabled only for contacts with Skype accounts. The menu item will be grayed out for SkypeOut contacts.
  - SKYPEOUT – menu item will be enabled only for SkypeOut contacts. The menu item will be grayed out for skypename contacts.
  - ALL – menu item will be enabled for all sorts of contacts, in which case you can basically omit this key altogether.

- CAPTION – Menu item text. Max 32 characters, enclose in quotes if the text contains whitespaces.
- HINT – Optional and currently unused.
- ICON – The directory path of the .PNG file of the menu icon. Maximum size of the icon is 32 × 32 pixels. Enclose in quotes. This parameter is optional.
- ENABLED – true | false – controls if the menu item is in enabled state. Menu item in disabled state remain visible in the menu but is grayed out and unclickable.
- ENABLE_MULTIPLE_CONTACTS – Optional and currently unused.

Version
Protocol 6 (API version 3.0)

DELETE MENU_ITEM

Removes a custom menu item. Note that custom menu items are removed automatically when the client application that created them is disconnected.

Syntax:
`DELETE MENU_ITEM <id>`

Example:

```
-> CREATE MENU_ITEM test01 CONTEXT contact CAPTION "TEST 01" ENABLED true
<- MENU_ITEM test01 CREATED
-> DELETE MENU_ITEM test01
<- DELETE MENU_ITEM test01
```

Version
Protocol 6 (API version 3.0)

SET MENU_ITEM

Syntax:
`SET MENU_ITEM <property> <value>`

This command enables you to change following properties of a custom menu item:

- CAPTION
- HINT
- ENABLED

Example:

```
-> CREATE MENU_ITEM test01 CONTEXT contact CAPTION "TEST 01" ENABLED true
<- MENU_ITEM test01 CREATED
-> SET MENU_ITEM test01 CAPTION "changed caption"
<- MENU_ITEM test01 CAPTION "changed caption"
```

Note that you can only change MENU_ITEM properties one at a time. To change both CAPTION and ENABLED properties of a MENU_ITEM, you will need two SET MENU_ITEM commands.

Version
Protocol 6 (API version 3.0)

MENU_ITEM click event

MENU_ITEM events are generated when a user clicks on a custom menu item. Note that each API client receives MENU_ITEM events only for menu items created from within their own code.

The message format is as follows:

```
<- MENU_ITEM <menu_id> CLICKED [<user_id>] CONTEXT <context>
[CONTEXT_ID <context_id>]
```

- The parameter is always returned as the ID of the menu item that was clicked.
- is only returned if the CONTEXT was either CONTACT, CALL or CHAT and contains the Skype name of the contact.
- is always returned as CONTEXT of the menu item.
- is only returned if the context was either CALL or CHAT. In case of a CALL, returns CALL ID, in case of a CHAT, it returns CHAT ID.

Example:

```
//-------------------------------------------------------------------------------
// Context = MYSELF
-> CREATE MENU_ITEM test05 CONTEXT MYSELF CAPTION "TEST" ENABLED true
// -- clicking --
<- MENU_ITEM test05 CLICKED CONTEXT myself
//-------------------------------------------------------------------------------
// Context = TOOLS
-> CREATE MENU_ITEM test06 CONTEXT TOOLS CAPTION "TEST" ENABLED true
// -- clicking --
<- MENU_ITEM test06 CLICKED CONTEXT tools
//-------------------------------------------------------------------------------
// Context = CONTACT
-> CREATE MENU_ITEM test07 CONTEXT CONTACT CAPTION "TEST" ENABLED true
// -- clicking --
<- MENU_ITEM test07 CLICKED echo123 CONTEXT contact
//-------------------------------------------------------------------------------
// Context = CALL
-> CREATE MENU_ITEM test03 CONTEXT CALL CAPTION "TEST" ENABLED true
// -- clicking --
<- MENU_ITEM test03 CLICKED echo123 CONTEXT call CONTEXT_ID 879
//-------------------------------------------------------------------------------
// Context = CHAT
-> CREATE MENU_ITEM test04 CONTEXT CHAT CAPTION "TEST" ENABLED true
// -- clicking --
<- MENU_ITEM test04 CLICKED echo123 CONTEXT chat CONTEXT_ID #tester/$echo123;559a71c0ef9d758b
```

Version
Protocol 6 (API version 3.0)

Skype Alert Events

Events, when created, appear in Skype UI on the right side of the mood message / profile panel as well as System Tray when Skype is in minimized state. Custom events can be created with the following API command:

```
-> CREATE EVENT <id> CAPTION <text> HINT <text>
```

Parameters:

- EVENT <id> – unique identifier, alphanumeric and must start with a letter.
- CAPTION <text> – displayed name of the menu item, enclosed in quotes if it contains whitespaces.
- HINT <text> – free-form text, enclosed in quotes if it contains whitespaces.

Custom events will be displayed on events tab as "Plugin messages". The CAPTION of the event will be displayed as a clickable link. Clicking on such link will generate a notification message in following format:

```
<- EVENT <id> CLICKED
```

Note that only the API client who created that particular event will receive such message.

The text given in `HINT` parameter will be displayed as hint, on mouse hover on the link.

Events remain in plugin message list as long as the API client that created them gets disconnected or are deleted from within API client code.

To delete events:
`-> DELETE EVENT <id>`

Example:

```
//---------------------------------------------------------------------
// Let there be a new event:
-> CREATE EVENT test1 CAPTION "Test message" HINT "Test message hint"
<- EVENT test1 CREATED
//---------------------------------------------------------------------
// At this point a red flag icon and "1 new event message" should appear
// on the mood message panel. Click on it, then click on "Test message".
// Following event is sent to your API client:
<- EVENT test1 CLICKED
//---------------------------------------------------------------------
// Clearing up the mess from event list
-> delete event test1
<- DELETE EVENT test1
```

Version
Protocol 6 (API version 3.0)

Application to application commands

The AP2AP feature in Skype allows two API clients to exchange information without the communication being visible on the client. Application to application communication has the following characteristics:

- From Skype version 3.1.0.150 it is no longer required for users to be in each other's contact list to be able to establish AP2AP connections. The only requirement for AP2AP connections is that both users have to be online.
- From API version 3.0, you can establish AP2AP connections between several instances of Skype that are logged in with the same skypename.
- Connections are only attempted to connectable users at CONNECT.
- Connections are established only when there is a matching application on the other side.
- When connection is dropped by one of the parties, all undelivered (stream) data will be lost. This is typically a problem when connection to remote aplication is dropped by a stream sender before receiver has acknowledged that it has received entire stream.
- The application name is limited to 32 bytes.
- Idle connections are dropped in a specific amount of time (typically 8 minutes).
- When connection is relayed, throttling is engaged.
- If the other party is logged in to multiple Skype instances, a stream for each instance is created.
- The stream write provides reliable transmission to deliver a large amount of data.
- The maximum amount of write to a stream can be 0xFFFF bytes long.
- Any character except 0×00 is allowed in a message.
- Datagrams are unreliable packets sent over a network (usually translates to UDP).
- The maximum size of datagrams is 1400 bytes.
- There is no guarantee that datagrams will be delivered.

Note: When connected to another user using application to application messaging, a user cannot install anything on the remote user's client without the express permission of the remote user.

Note on AP2AP streams:
With Skype4Com library versions prior to 1.0.28, re-entrant event handlers caused stream packets to be retrieved from receiving side in incorrect order. If you experience problems with packet order (and you are using Skype4Com library), make sure you upgrade it to version 1.0.28.

Another note on Skype4Com library:
Binary data transfers via ap2ap functionality of Skype4Com library can sometimes lead to data getting partially scrambled. To make sure your binary data is transmitted properly, we strongly suggest that you use base64 encoding to convert your data to strings before passing those strings to Skype4Com IApplication.!SendDatagram and IApplicationStream.Write methods.

The reason for this phenomenon is that due to how string parameters are handled when communicating with ActiveX objects, all Skype API commands that are passed to or retrieved from the Skype client by Skype4Com library are passed through UTF-8 encoding routine. This includes commands dealing with application to application datagrams and stream writes/reads. Those UTF-8 encoding routines occasionally produce different results, depending on additional language packs a user has installed in Windows.

For code example on base64 encoding/decoding algorithms, refer to A2AStreams.pas example linked below.

Read an [#COMMAND_AP2AP_EXAMPLE application to application example] to get you started.

Skype4Com example:

- A2AChat.pas
- A2AStreams.pas

AP2AP CREATE

This command registers a new application object with Skype. Application name cannot contain whitespaces.

Syntax:
```
CREATE APPLICATION <appname>
```

Response
If successful, the command is echoed back

Parameters:
<appname> : An arbitrary name to identify the application that will be exchanging data

Errors

- ERROR 536 CREATE: no object or type given
- ERROR 537 CREATE: Unknown object type given
- ERROR 540 CREATE APPLICATION: Missing or invalid name
- ERROR 541 APPLICATION: operation failed - in case an application with this name already exists

Version

- Protocol 5
- Skype for Windows 1.4

AP2AP CONNECT

This command creates a stream from the application to another Skype user's instance of the same application.

Syntax:
```
ALTER APPLICATION <appname> CONNECT <skypename>
```

Response:
If successful, the command is echoed back

Example 1: no matching application on the other side

```
-> #ID1 alter application test connect testuser
<- #ID1 ALTER APPLICATION test CONNECT testuser
<- APPLICATION test CONNECTING testuser
<- APPLICATION test CONNECTING
```

Note that only the initial feedback notification is echoed back with command ID.

Example 2: Matching application on remote was found

```
//-------------------------------------------------------
// From initiator perspective
-> #ID1 alter application test connect anappo2
<- #ID1 ALTER APPLICATION test CONNECT anappo2
<- APPLICATION test CONNECTING anappo2
<- APPLICATION test CONNECTING
<- APPLICATION test STREAMS anappo2:1
//-------------------------------------------------------
// From remote perspective
<- APPLICATION test STREAMS anappo:1
```

Parameters:

- <appname> : An arbitary name to identify the application that will be exchanging data
- <skypename> : The user to connect to this application

Errors:

- ERROR 546 ALTER APPLICATION: Missing or invalid action
- ERROR 547 ALTER APPLICATION CONNECT: Invalid user handle

Version:

- Protocol 5
- Skype for Windows 1.4

Note:

If the user identified by <skypename> is logged in from multiple locations, a stream will be created to each location.

AP2AP WRITE

This command writes text into the application stream identified by the destination user's Skypename and stream ID.

Syntax

```
ALTER APPLICATION <appname> WRITE <skypename>:<id> <text>
```

Response

If successful, the command is echoed back
Note: There is a bug in Skype 1.4 where, following an application WRITE event, Skype reports that the number of bytes sent is two characters greater than that which is actually written.

Parameters

- <appname> : An arbitary name to identify the application that will be exchanging data
- <skypename> : The name of the skype contact to whom the message will be sent
- <id> : The numeric identifier for the skype instance to which the message will be sent
- <text> : The text to send

Errors

- ERROR 546 ALTER APPLICATION : Missing or invalid action
- ERROR 550 ALTER APPLICATION READ: Missing or Invalid stream identifier

Version

- Protocol 5
- Skype for Windows 1.4

Example

```
//Send "Hello world!" to user "testtest20" stream "1" (application "exe")
-> ALTER APPLICATION exe WRITE testtest20:1 Hello world!
<- ALTER APPLICATION exe WRITE testtest20:1
// The message has been queued for sending, message length is reported back
<- APPLICATION exe SENDING testtest20:1 14
// The message has been sent -- note missing stream ID from the end of response
<- APPLICATION exe SENDING
-> ALTER APPLICATION exe WRITE testtest20:1 1234567890
<- ALTER APPLICATION exe WRITE testtest20:1
<- APPLICATION exe SENDING testtest20:1 12
<- APPLICATION exe SENDING
```

AP2AP DATAGRAM

This command sends a datagram to the application stream.

Syntax:
`ALTER APPLICATION <appname> DATAGRAM <skypename>:<id> <text>`

Parameters

- <appname> : An arbitrary name to identify the application that will be exchanging data
- <skypename> : skypename of the remote party
- <id> : stream ID
- <text> : datagram content (0×00 is not allowed, so to use this for binary transfers you need to convert the data to remove nulls, using base64 or base128 for example).

Example:

```
//-------------------------------------------------------
// Creating and connecting application
// (from sender perspective)
-> CREATE APPLICATION test
<- CREATE APPLICATION test
-> ALTER APPLICATION test CONNECT anappo
<- ALTER APPLICATION test CONNECT anappo
<- APPLICATION test CONNECTING anappo
//-------------------------------------------------------
// Note that a STREAMS event notification is
// generated automatically upon connect.
<- APPLICATION test STREAMS anappo:1
//-------------------------------------------------------
// Sending datagram
-> ALTER APPLICATION test DATAGRAM anappo:1 BBBBBBBBBBBBB
<- ALTER APPLICATION test DATAGRAM anappo:1
//-------------------------------------------------------
// Following notification contains the number of
// characters in datagram
<- APPLICATION test SENDING anappo:1=14
<- APPLICATION test SENDING

//-------------------------------------------------------
// Same thing from receiver perspective
-> CREATE APPLICATION test
<- CREATE APPLICATION test
<- APPLICATION test STREAMS anappo2:1
//-------------------------------------------------------
// Note that receiver does not get a separate notification
// with size of received datagram.
<- APPLICATION test DATAGRAM anappo2:1 BBBBBBBBBBBBB
```

Errors

- ERROR 546 ALTER APPLICATION : Missing or invalid action

- ERROR 551 ALTER APPLICATION DATAGRAM: Missing or invalid stream identifier

Version

- Protocol 5
- Skype for Windows 1.4

AP2AP READ

This command reads data from an application stream.

Syntax
`ALTER APPLICATION <appname> READ <skypename>:<id>`

Response
If successful, the command is echoed back

Parameters

- <appname> : An arbitrary name to identify the application that will be exchanging data
- <skypename> : skypename of the remote party
- <id> : stream ID

Example:

```
//-------------------------------------------------------
// Sender
-> CREATE APPLICATION test
<- CREATE APPLICATION test
-> ALTER APPLICATION test CONNECT anappo
<- ALTER APPLICATION test CONNECT anappo
<- APPLICATION test CONNECTING anappo
<- APPLICATION test CONNECTING
<- APPLICATION test STREAMS anappo:1
-> ALTER APPLICATION test WRITE anappo:1 AAAAAA
<- ALTER APPLICATION test WRITE anappo:1
<- APPLICATION test SENDING anappo:1=8
<- APPLICATION test SENDING

//-------------------------------------------------------
// Receiver
-> CREATE APPLICATION test
<- CREATE APPLICATION test
//-------------------------------------------------------
// Streams notification we received on remote connect
<- APPLICATION test STREAMS anappo2:1
//-------------------------------------------------------
// Packet notification message including packet size
<- APPLICATION test RECEIVED anappo2:1=6
//-------------------------------------------------------
// Reading the packet
-> ALTER APPLICATION test READ anappo2:1
<- ALTER APPLICATION test READ anappo2:1 AAAAAA
<- APPLICATION test RECEIVED
```

Errors:

- ERROR 546 ALTER APPLICATION : Missing or invalid action
- ERROR 550 ALTER APPLICATION READ: Missing or invalid stream identifier

Version

- Protocol 5
- Skype for Windows 1.4

AP2AP DISCONNECT

This command disconnects a user stream from an application.

Syntax
```
ALTER APPLICATION <appname> DISCONNECT <skypname>:<id>
```

Response
If successful, the command is echoed back

Parameters

- <appname> : An arbitrary name to identify the application that will be exchanging data
- <skypename>:<id> : The user and stream to disconnect

Example:

```
//-------------------------------------------------------
// From initiator perspective
-> #ID2 alter application test disconnect anappo2:1
<- #ID2 ALTER APPLICATION test DISCONNECT anappo2:1
<- APPLICATION test STREAMS
//-------------------------------------------------------
// From remote perspective
<- APPLICATION test STREAMS
```

Note that if you use re-connect to the same remote user after disconnecting, the part of the streams notification will increment itself.

```
-> ALTER APPLICATION test CONNECT anappo2
<- ALTER APPLICATION test CONNECT anappo2
<- APPLICATION test CONNECTING anappo2
<- APPLICATION test CONNECTING
<- APPLICATION test STREAMS anappo2:2
```

Errors:

- ERROR 546 ALTER APPLICATION : Missing or invalid action
- ERROR 548 ALTER APPLICATION DISCONNECT: Invalid stream identifier

Version

- Protocol 5
- Skype for Windows 1.4

AP2AP DELETE

This command deletes an application and drops all connections to it.

Syntax
```
DELETE APPLICATION <appname>
```

Response
If successful, the command is echoed back

Parameters
<appname> : The name of the application to be deleted

NB! If your application had open connections to remote users, these users will not receive notifications when you delete that application.

Errors:

- ERROR 538 DELETE: no object or type given
- ERROR 539 DELETE: Unknown object type given
- ERROR 542 DELETE APPLICATION : missing or invalid application name
- ERROR 541 APPLICATION: operation failed

Version

- Protocol 5
- Skype for Windows 1.4

Application to application example

Jim and Joe are two users who installed “toru” application.

```
// register application on both sides
[JIM] => CREATE APPLICATION toru
[JIM] <= CREATE APPLICATION toru
[JOE] => CREATE APPLICATION toru
[JOE] <= CREATE APPLICATION toru
// JIM initiates communication to JOE
[JIM] => ALTER APPLICATION toru CONNECT joe
[JIM] <= ALTER APPLICATION toru CONNECT joe
// connection establishing ...
[JIM] <= APPLICATION toru CONNECTING joe
// .. and is successful
[JIM] <= APPLICATION toru CONNECTING
// .. and creates one stream
[JIM] <= APPLICATION toru STREAMS joe:1
// and JOE is notified by new stream
[JOE] <= APPLICATION toru STREAMS jim:1
// JIM sends data over stream to JOE
[JIM] => ALTER APPLICATION toru WRITE joe:1 Hello world!
[JIM] <= ALTER APPLICATION toru WRITE joe:1
// stay tuned while data is transmitted...
[JIM] <= APPLICATION toru SENDING joe:1
// .. and you are notified on delivery success
[JIM] <= APPLICATION toru SENDING
// JOE receives notification about the incoming message
[JOE] <= APPLICATION toru RECEIVED jim:1
// .. and reads data from stream
[JOE] => ALTER APPLICATION toru READ jim:1
[JOE] <= ALTER APPLICATION toru READ jim:1 Hello world!
// ... and is notified that stream is empty
[JOE] <= APPLICATION toru RECEIVED
// JOE sends back acknowledgement of message
// A datagram is used because it is not so important to acknowledge
[JOE] => ALTER APPLICATION toru DATAGRAM jim:1 Hello back!
[JOE] <= ALTER APPLICATION toru DATAGRAM jim:1
// Now data is transmitted...
[JOE] <= APPLICATION toru SENDING jim:1=11
// .. and notificed when it was sent (but delivery not assured)
[JOE] <= APPLICATION toru SENDING
// JIM receives datagram notifcation
[JIM] <= APPLICATION toru DATAGRAM joe:1 Hello back!
// JIM decides to end the communication
[JIM] => ALTER APPLICATION toru DISCONNECT joe:1
[JIM] <= ALTER APPLICATION toru DISCONNECT joe:1
// .. and when stream is closed it is notified
[JIM] <= APPLICATION toru STREAMS
// Also JOE receives notification that stream was closed
[JOE] <= APPLICATION toru STREAMS
// JIM unregisters applicaton
[JIM] => DELETE APPLICATION toru
[JIM] <= DELETE APPLICATION toru
// JOE unregisters applicaton
[JOE] => DELETE APPLICATION toru
[JOE] <= DELETE APPLICATION toru
```

Voice Streams

Refer to CALL object for properties relevant to manipulating voice streams.

To change voice stream properties of a CALL object, there are three extensions of the ALTER CALL command:

```
ALTER CALL <id> SET_INPUT SOUNDCARD="default" | PORT="port_no" |
FILE="FILE_LOCATION"
```

This enables you to set a port or a wav file as a source of your voice, instead of a microphone.

```
ALTER CALL <id> SET_OUTPUT SOUNDCARD="default" | PORT="port_no" |
FILE="FILE_LOCATION"
```

Redirects incoming transmission to a port or a wav file.

```
ALTER CALL <id> SET_CAPTURE_MIC PORT="port_no" |
FILE="FILE_LOCATION"
```

Captures your own voice from microphone to a port or a wav file.

Note that as of version 3.5.0.202 redirecting of voice streams is also available for voicemails. Look for corresponding ALTER commands at the end of this section.

Example 1 – capturing incoming transmission

```
//-------------------------------------------------------------
// In this example we will call Skype call testing service
// and play around with redirecting inputs and outputs.
// First, lets try capturing incoming transmission into a file.
-> call echo123
<- CALL 808 STATUS UNPLACED
<- CALL 808 STATUS ROUTING
<- CALL 808 STATUS RINGING
<- CONTACTS FOCUSED
<- CALL 808 VAA_INPUT_STATUS FALSE
<- CALL 808 STATUS INPROGRESS
//-------------------------------------------------------------
// Ok, the call is now in progress and the helpful lady robot
// on the other side is talking. We can capture her voice to
// a wav file by issuing the following command:
-> ALTER CALL 808 SET_OUTPUT file="c:\test.wav"
<- ALTER CALL 808 SET_OUTPUT file="c:\test.wav"
<- CALL 808 STATUS FINISHED
//-------------------------------------------------------------
// We now have a c:\test.wav file, containing the incoming transmission.
```

Example 2 – altering the source of the outgoing transmission

```
//-------------------------------------------------------------
// Let's call the helpful robot again and play a little trick on her.
// By altering sound input source, we can send her back her own voice
// that we recorded in our previous example.
-> call echo123
<- CALL 846 STATUS UNPLACED
<- CALL 846 STATUS ROUTING
<- CALL 846 STATUS RINGING
<- CALL 846 VAA_INPUT_STATUS FALSE
<- CALL 846 STATUS INPROGRESS
//-------------------------------------------------------------
// Wait until the lady robot asks for you to speak, then set
// call input to a file instead of microphone.
-> ALTER CALL 846 SET_INPUT file="c:\test.wav"
<- ALTER CALL 846 SET_INPUT file="c:\test.wav"
<- CALL 846 VAA_INPUT_STATUS TRUE
<- CALL 846 VAA_INPUT_STATUS FALSE
//-------------------------------------------------------------
// If the sound from fail was sent correctly, you should hear
// the robot's voice in the playback phase of the call test.
<- CALL 846 STATUS FINISHED
```

Example 3 – capturing voice from the microphone

```
//-------------------------------------------------------------
// In this example, we will capture our own voice.
```

```
<- CALL 889 STATUS UNPLACED
<- CALL 889 STATUS ROUTING
<- CALL 889 STATUS RINGING
<- CONTACTS FOCUSED
<- CALL 889 VAA_INPUT_STATUS FALSE
<- CALL 889 STATUS INPROGRESS
//-------------------------------------------------------------
// Wait until the lady robot asks you to speak, then switch on
// sound capture to a file and talk.
-> ALTER CALL 889 SET_CAPTURE_MIC file="c:\test.wav"
<- ALTER CALL 889 SET_CAPTURE_MIC file="c:\test.wav"
<- CALL 889 STATUS FINISHED
//-------------------------------------------------------------
// The test.wav file should now contain your own voice.
```

The relevant properties of a CALL object can be accessed in a following manner:

```
-> GET CALL 748 INPUT
<- CALL 748 INPUT SOUNDCARD="default"
-> GET CALL 748 OUTPUT
<- CALL 748 OUTPUT SOUNDCARD="default"
-> GET CALL 748 VAA_INPUT_STATUS
<- CALL 748 VAA_INPUT_STATUS FALSE
```

Audio format
File: WAV PCM

Sockets: raw PCM samples

1. KHz mono, 16 bit

Note:
The voice access API works with virtual audio cables (VACs) versions 3 and 4. However, users with VAC version 3.x may encounter distorted sound for the initial one or two seconds of a call.

When you redirect a port, Skype acts as a TCP client and goes looking for a TCP server on the given port. To receive voice stream data, you have to have your own TCP server running on that port.

Skype4Com example:

- VoiceStreams.pas – Example of how to redirect voice streams to a TCP port.
- VoiceMail2WAV.pas – Example on how to save voicemails into WAV files.
- VoiceMail2Port.pas – Example on how to redirect voicemail output to a TCP port.

Version
Skype API version 2.6 (protocol 6)

ALTER CALL SET_INPUT

This enables you to set a port or a wav file as a source of your voice, instead of a microphone.

Syntax:
```
-> ALTER CALL <id> SET_INPUT SOUNDCARD="default" | PORT="port_no" |
FILE="FILE_LOCATION"

<- ALTER CALL <id> SET_INPUT SOUNDCARD="default" | PORT="port_no" |
FILE="FILE_LOCATION"
```

Note that for now, the SOUNDCARD parameter only accepts one value – "default".

Example:

```
-> ALTER CALL 846 SET_INPUT file="c:\test.wav"
<- ALTER CALL 846 SET_INPUT file="c:\test.wav"
```

Version
Skype API version 2.6 (protocol 6)

ALTER CALL SET_OUTPUT

This command redirects incoming transmission to a port or a wav file.

Syntax:

```
-> ALTER CALL <id> SET_OUTPUT SOUNDCARD="default" | PORT="port_no" |
FILE="FILE_LOCATION"

<- ALTER CALL <id> SET_OUTPUT SOUNDCARD="default" | PORT="port_no" |
FILE="FILE_LOCATION"
```

Note that for now, the SOUNDCARD parameter only accepts one value – "default". If this parameter is omitted or differs from "default", the soundcard output is muted.

Example:

```
-> ALTER CALL 808 SET_OUTPUT file="c:\test.wav"
<- ALTER CALL 808 SET_OUTPUT file="c:\test.wav"
```

Version
Skype API version 2.6 (protocol 6)

ALTER CALL SET_CAPTURE_MIC

This command captures your own voice from microphone to a port or a wav file.

Syntax:

```
-> ALTER CALL <id> SET_CAPTURE_MIC PORT="port_no" |
FILE="FILE_LOCATION"

<- ALTER CALL <id> SET_CAPTURE_MIC PORT="port_no" |
FILE="FILE_LOCATION"
```

Example:

```
-> ALTER CALL 889 SET_CAPTURE_MIC file="c:\test.wav"
<- ALTER CALL 889 SET_CAPTURE_MIC file="c:\test.wav"
```

Version
Skype API version 2.6 (protocol 6)

ALTER VOICEMAIL SET_INPUT

This enables you to set a port or a wav file as a source of voicemail's input instead of a microphone.

Syntax:

```
-> ALTER VOICEMAIL <id> SET_INPUT SOUNDCARD="default" |
PORT="port_no" | FILE="FILE_LOCATION"

<- ALTER VOICEMAIL <id> SET_INPUT SOUNDCARD="default" |
PORT="port_no" | FILE="FILE_LOCATION"
```

Note that for now, the SOUNDCARD parameter only accepts one value – "default". If this parameter is omitted or differs from "default", the soundcard input is muted.

Example:

```
 > ALTER VOICEMAIL 146 SET_INPUT file="c:\test.wav"
<- ALTER VOICEMAIL 146 SET_INPUT file="c:\test.wav"
```

Version
Skype API version 3.5.0.202 (protocol 8)

ALTER VOICEMAIL SET_OUTPUT

This command redirects voicemail output to a port or a wav file.

Syntax:

```
-> ALTER VOICEMAIL <id> SET_OUTPUT SOUNDCARD="default" |
PORT="port_no" | FILE="FILE_LOCATION"

<- ALTER VOICEMAIL <id> SET_OUTPUT SOUNDCARD="default" |
PORT="port_no" | FILE="FILE_LOCATION"
```

Note that for now, the SOUNDCARD parameter only accepts one value – "default". If this parameter is omitted or differs from "default", the soundcard output is muted.

Example:

```
-> ALTER VOICEMAIL 108 SET_OUTPUT file="c:\test.wav"
<- ALTER VOICEMAIL 108 SET_OUTPUT file="c:\test.wav"
```

Version
Skype API version 3.5.0.202 (protocol 8)

ALTER VOICEMAIL SET_CAPTURE_MIC

This command captures your own voice from microphone to a port or a wav file.

Syntax:

```
-> ALTER VOICEMAIL <id> SET_CAPTURE_MIC PORT="port_no" |
FILE="FILE_LOCATION"

<- ALTER VOICEMAIL <id> SET_CAPTURE_MIC PORT="port_no" |
FILE="FILE_LOCATION"
```

Example:

```
-> ALTER VOICEMAIL 189 SET_CAPTURE_MIC file="c:\test.wav"
<- ALTER VOICEMAIL 189 SET_CAPTURE_MIC file="c:\test.wav"
```

Version
Skype API version 3.5.0.202 (protocol 8)

Testing connections

This command can be used to test whether connection between your application and Skype is still alive. This command is not meant to query online status of remote users.

Syntax
PING

Response
If successful PONG is echoed back

Version
Protocol 1
*

Note that from protocol 6 and onward, the `PONG` reply to `PING` is asynchronous.

Objects

This section contains the Skype objects.

USER object

NB! When you retrieve `USER` object records with `SEARCH USERS` command, the user profile data is guaranteed to be accessible with `GET USER <user_id> <property_name>` commands **only** until another `SEARCH` command is executed. The reason for this is that big `SEARCH` commands can and often trigger Skype's internal garbage collection routine that can clear out the data retreived by previous searches.

The user object has the following properties:

- `HANDLE` – username, for example: `USER pamela HANDLE pamela` .
- `FULLNAME` – user's full name, for example: `USER pamela FULLNAME Jane Doe` .
- `BIRTHDAY` – user's birth date in YYYYMMDD format, for example: `USER bitman BIRTHDAY 19780329` .
- `SEX` – example: `USER pamela SEX UNKNOWN` . Values:
  - `UNKNOWN` – user has not specified sex in personal profile.
  - `MALE`
  - `FEMALE`

- `LANGUAGE` – name of language, for example: `USER mike LANGUAGE English` . In protocol 4 with the ISO 639 prefix, example: `USER mike LANGUAGE en English` .
- `COUNTRY` – name of country, for example: `USER mike COUNTRY Estonia` . In protocol 4 with the ISO 3166 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` – always returns `TRUE` . Example: `USER pamela HASCALLEQUIPMENT TRUE` .
- `IS_VIDEO_CAPABLE` – possible values: `True` or `False`
- `IS_VOICEMAIL_CAPABLE` – possible values: `True` or `False`
- `BUDDYSTATUS` – example: `USER pamela BUDDYSTATUS 2` . Possible `BUDDYSTATUS` values:
  - `0` – never been in contact list.
  - `1` – deleted from contact list. (read-write)
  - `2` – pending authorisation. (read-write)
  - `3` – added to contact list.

- `ISAUTHORIZED` – (read-write) is user authorized by current user? Example: `USER pamela ISAUTHORIZED TRUE` . Values:
  - `TRUE`
  - `FALSE`

- `ISBLOCKED` – (read-write) is user blocked by current user? Example: `USER spammer ISBLOCKED TRUE` . Values:
  - `TRUE`
  - `FALSE`

- `ONLINESTATUS` – user online status, for example: `USER mike ONLINESTATUS ONLINE` . Possible 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 1078999979 .
- CAN_LEAVE_VM – is it possible to send voicemail to user? Example: USER test CAN_LEAVE_VM TRUE . Possible values:
  - TRUE
  - FALSE

- SPEEDDIAL – (read-write) speeddial code assigned to user.
- RECEIVEDAUTHREQUEST – text message for authorization request; available only when user asks for authorization.
- MOOD_TEXT – mood text for user (mood text is only visible to authorised users; visible in Skype for Windows 2.0).
- RICH_MOOD_TEXT – advanced version of user's mood message. See SET PROFILE RICH_MOOD_TEXT command for more information. Introduced in API version 3.0
- ALIASES <text> – list of assigned aliases (aliases are only visible as a result of a direct match for alias search).
- TIMEZONE <offset> – time offset from GMT in minutes; visible in Skype for Windows 2.0.
- IS_CF_ACTIVE – whether the user has Call Forwarding activated or not. Possible values:
  - TRUE
  - FALSE

- NROF_AUTHED_BUDDIES – Stores the number of authorized contacts in the contact list.

Most user properties are read-only. The following properties are read-write and can be modified with the SET command:

- BUDDYSTATUS
  - 1 – delete from buddylist
  - 2 – add user into contactlist and ask for authorization: SET USER echo123 BUDDYSTATUS 2 Please authorize me

- ISBLOCKED
  - TRUE – block user
  - FALSE – unblock user

- ISAUTHORIZED
  - TRUE – authorize user
  - FALSE – dismiss authorization for user

- SPEEDDIAL – speeddial code assigned to user
- DISPLAYNAME – By default this property is empty. If a value is assigned to this property with SET <skypename> DISPLAYNAME <value> then that value will be displayed in Skype UI instead of user's FULLNAME.

PROFILE object

Use the GET PROFILE command to retrieve profile information. The PROFILE object has the following properties:

- PSTN_BALANCE – (**read only**) SkypeOut balance value. Note that the precision of profile balance value is currently fixed at 2 decimal places, regardless of currency or any other settings.
- PSTN_BALANCE_CURRENCY – (**read only**) SkypeOut currency value
- FULLNAME – text
- BIRTHDAY – yyyymmdd, 0 is returned if not set; no partial birthday allowed
- SEX – MALE | FEMALE | UNKNOWN
- LANGUAGES – [lang[lang]*] — lang is a two letter ISO code (en, de, et)
- COUNTRY – iso2 name, a two letter ISO code; name – country name
- IPCOUNTRY – GeoIP location, country code in two letter ISO format
- PROVINCE – text
- CITY – text
- PHONE_HOME – text
- PHONE_OFFICE – text
- PHONE_MOBILE – text
- HOMEPAGE – text
- ABOUT – text
- MOOD_TEXT – user's mood message (the plain text version).
- RICH_MOOD_TEXT – advanced version of user's mood message. See SET PROFILE RICH_MOOD_TEXT command for more information. Introduced in API version 3.0
- TIMEZONE – Offset is given in seconds, according to this formula: (24 + offset_from _GMT) * 3600. For example, value of this property for Estonia (GMT+2) would be 93600 (26*3600).
- CALL_APPLY_CF – To enable/disable call forwarding – See Call forwarding

- CALL_NOANSWER_TIMEOUT – Time out on call – See Call forwarding
- CALL_FORWARD_RULES – See Call forwarding
- CALL_SEND_TO_VM – To enable/disable voicemail for forwarded calls – See Call forwarding
- SMS_VALIDATED_NUMBERS – A read-only property that contains a comma-separated list of phone numbers the user has registered for usage in reply-to field of SMS messages. See [#SMS_NUMBER_VALIDATION Setting mobile phone number on reply-to field in outgoing SMS messages] section for further information.

CALL object

The CALL object has the following properties:

- TIMESTAMP – time when call was placed (UNIX timestamp), for example `CALL 17 TIMESTAMP 1078958218`
- PARTNER_HANDLE – for example `CALL 17 PARTNER_HANDLE mike`. In case of SkypeOut and SkypeIn calls this property contains the PSTN number of remote party, prefixed by countrycode (+123456789).
- PARTNER_DISPNAME – for example `CALL 17 PARTNER_DISPNAME Mike Mann`
- TARGET_IDENTITY – This property is set when you a) have a SkypeIn number and b) receive an incoming PSTN call. The value of call's target identity property is then set to your own SkypeIn number. This property is not set if the incoming call is P2P. This property was introduced in API version 3.1
- CONF_ID – if the `CONF_ID>0` the call is a conference call, for example: `CALL 17 CONF_ID 0`
- TYPE – call type, for example: `CALL 17 TYPE OUTGOING_PSTN` . Possible 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, for example: `CALL 17 STATUS FAILED` . Possible values:
  - UNPLACED – call was never placed
  - ROUTING – call is currently being routed
  - EARLYMEDIA – with pstn it is possible that before a call is established, early media is played. For example it can be a calling tone or a waiting message such as all operators are busy.
  - FAILED – call failed – try to get a 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
  - CANCELLED (Protocol 2)
  - TRANSFERRING – Refer to ALTER CALL TRANSFER command. Added in protocol 7 (API version 3.0)
  - TRANSFERRED – Refer to ALTER CALL TRANSFER command. Added in protocol 7 (API version 3.0)
  - VM_BUFFERING_GREETING – voicemail greeting is being downloaded
  - VM_PLAYING_GREETING – voicemail greeting is being played
  - VM_RECORDING – voicemail is being recorded
  - VM_UPLOADING – voicemail recording is finished and uploaded into server
  - VM_SENT – voicemail has successfully been sent
  - VM_CANCELLED – leaving voicemail has been cancelled
  - VM_FAILED – leaving voicemail failed; check FAILUREREASON
  - WAITING_REDIAL_COMMAND – This status is set when your outgoing call to PSTN gets rejected by remote party. This state was added in version 3.5 (protocol 8).
  - REDIAL_PENDING – This status is set when you press redial button on the Call Phones tab of the Skype interface. This state was added in version 3.5 (protocol 8).

- VIDEO_STATUS – Commands ALTER CALL VIDEO_SEND and RECEIVE ALTER CALL VIDEO_ RECEIVE can be used to change call video status. Possible values of this property are:
  - VIDEO_NONE
  - VIDEO_SEND_ENABLED
  - VIDEO_RECV_ENABLED
  - VIDEO_BOTH_ENABLED

- VIDEO_SEND_STATUS and VIDEO_RECEIVE_STATUS – possible values of this property are:
  - NOT_AVAILABLE – the client does not have video capability because video is disabled or a webcam is unplugged).
  - AVAILABLE – the client is video-capable but the video is not running (can occur during a manual send).
  - STARTING – the video is sending but is not yet running at full speed.
  - REJECTED – the receiver rejects the video feed (can occur during a manual receive).
  - RUNNING – the video is actively running.
  - STOPPING – the active video is in the process of stopping but has not halted yet.

- `PAUSED` – the video call is placed on hold.

- `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 the case of a PSTN call, for example: `CALL 26 PSTN_STATUS 6500 PSTN connection creation timeout` .
- `CONF_PARTICIPANTS_COUNT` – number of non-hosts in the case of a conference call. Possible values are:
  - `0` – call is not a conference. For the host, `CONF_PARTICIPANTS_COUNT` is always 0.
  - `1` – call is a former conference.
  - `2, 3, 4` – call is a conference. Note that from 2.5 and upwards, Skype API manages conference participation in a slightly different manner. In newer versions, after the call is finished, the `CONF_PARTICIPANTS_COUNT` reports highest number of participants the call had at any given time.

- `CONF_PARTICIPANT n` – the username of the nth participant in a conference call, the call type and status and the displayname of participants who are not the host. For example: `CALL 59 CONF_PARTICIPANT 1 echo123 INCOMING_P2P INPROGRESS Echo Test Service` .
- `VM_DURATION`
- `VM_ALLOWED_DURATION` – maximum duration in seconds allowed to leave voicemail
- `RATE` – expressed as cost per minute (added in protocol 6).
- `RATE_CURRENCY` – EUR|USD.. This property gets populated from currency selected in Skype account details – `PSTN_BALANCE_CURRENCY` property of the `PROFILE` object. However, the value of `PSTN_BALANCE_CURRENCY` can change in time (added in protocol 6).
- `RATE_PRECISION` – the number of times to divide `RATE` by 10 to get the full currency unit. For example, a `RATE` of 1234 with `RATE_PRECISION` of 2 amounts to 12.34 (added in protocol 6).
- `INPUT` – New in API version 2.6 Refer to Voice Streams section for more information. Can have following values:
  - `SOUNDCARD="default"` – default is currently the only acceptable value.
  - `PORT="port_no"` – the ID of the audio port (1..65535)
  - `FILE="filename.wav"` – the path and name of the audio file.

- `OUTPUT` – can have all the same values as `INPUT` property. Refer to Voice Streams section for more information. New in API version 2.6
- `CAPTURE_MIC` – can have all the same values as `INPUT` and `OUTPUT` properties. Refer to Voice Streams section for more information. New in API version 2.6
- `VAA_INPUT_STATUS` – true|false, indicates if voice input is enabled. New in API version 2.6
- `FORWARDED_BY` – Contains identity of the user who forwarded a call. If the user who forwarded the call could not be identified then this property will be set to "?". New in API version 2.6
- `TRANSFER_ACTIVE` – Refer to ALTER CALL TRANSFER command. Added in protocol 7 (API version 3.0)
- `TRANSFER_STATUS` – Refer to ALTER CALL TRANSFER command. Added in protocol 7 (API version 3.0)
- `TRANSFERRED_BY` – Refer to ALTER CALL TRANSFER command. Added in protocol 7 (API version 3.0)
- `TRANSFERRED_TO` – Refer to ALTER CALL TRANSFER command. Added in protocol 7 (API version 3.0)

Notes

- Status values for voicemails (`VM_xxx`) and `VM_DURATION/VM_ALLOWED_DURATION` apply to calls which are forwarded into voicemail. This feature was introduced in protocol 5.
  Most call properties are read-only. The following properties are read-write and can be modified with the `SET` command:

- `STATUS` – for call control. Possible values:
  - `ONHOLD` – hold call
  - `INPROGRESS` – answer or resume call
  - `FINISHED` – hang up call

- `SEEN` – sets call as seen, so that a missed call is seen and can be removed from the 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 the ID of another call.

MESSAGE object

Version

Protocol 1, **deprecated in protocol 3** and replaced by the CHATMESSAGE object.

Properties

- `TIMESTAMP` – time when the message was sent (UNIX timestamp), for example: `MESSAGE 21 TIMESTAMP 1078958218`
- `PARTNER_HANDLE` – for example `MESSAGE 21 PARTNER_HANDLE mike`

- PARTNER_DISPNAME – for example MESSAGE 21 PARTNER_DISPNAME Mike Mann
- CONF_ID – not used.
- TYPE – message type, for example MESSAGE 21 TYPE TEXT . Possible TYPE values:
  - AUTHREQUEST – authorization request
  - TEXT – IM or topic set
  - CONTACTS – contacts data
  - UNKNOWN – other

- STATUS – message status, for example MESSAGE 21 STATUS QUEUED . Possible values:
  - SENDING – message is being sent
  - SENT – message was sent
  - FAILED – message sending failed. Try to get a FAILUREREASON for more information.
  - RECEIVED – message has been received
  - READ – message has been read
  - IGNORED – message was ignored
  - QUEUED – message is queued

- FAILUREREASON – for example MESSAGE 21 FAILUREREASON 1 (numeric).
- BODY – message body, for example MESSAGE 21 BODY Hi, what's up?
  Most message properties are read-only. The following property is read-write and can be modified with the SET command:
- SEEN – the message is seen and will be removed from missed messages list. The UI sets this automatically if auto-popup is enabled for the user.

CHAT object

Version

Protocol 3 (updated in protocol 7)

Properties

- NAME – chat ID, for example CHAT #test_l/$6a072ce5537c4044 NAME #test_l/$6a072ce5537c4044
- TIMESTAMP – time when chat was created, for example CHAT #test_l/$6a072ce5537c4044 TIMESTAMP 1078958218
- ADDER – user who added the current user to chat, for example CHAT 1078958218 ADDER k6rberebane
- STATUS – chat status, for example CHAT #test_l/$6a072ce5537c4044 STATUS MULTI_SUBSCRIBED .
  Possible values:
  - LEGACY_DIALOG – old style IM
  - DIALOG – 1:1 chat.
  - MULTI_SUBSCRIBED – participant in chat
  - UNSUBSCRIBED – left chat

- POSTERS – members who have posted messages, for example CHAT #test_l/$6a072ce5537c4044 POSTERS k6rberebane test_3
- MEMBERS – all users who have been there, for example CHAT #test_l/$6a072ce5537c4044 MEMBERS k6rberebane test test_2 test_3
- TOPIC – chat topic. Example: CHAT #test_l/$6a072ce5537c4044 TOPIC API testimine
- TOPICXML – set when a chat topic contains XML formatting elements (topic was changed with ALTER CHAT SETTOPICXML command) This property works in parallel with TOPIC property – when TOPICXML is set, the value is stripped of XML tags and updated in TOPIC.
- CHATMESSAGES – all messages IDs in this chat, for example CHAT #test_l/$6a072ce5537c4044 CHATMESSAGES 34, 35, 36, 38, 39
- ACTIVEMEMBERS – members who have stayed in chat, for example CHAT #test_l/$6a072ce5537c4044 ACTIVEMEMBERS k6rberebane test_2 test_3
- FRIENDLYNAME – name shown in chat window title, for example CHAT #test_l/$6a072ce5537c4044 FRIENDLYNAME Test Test XX | tere ise ka
- CHATMESSAGES – list of chatmessage identifiers
- RECENTCHATMESSAGES – list of missed/recent chatmessage identifiers
- BOOKMARKED – TRUE|FALSE (added in protocol version 6 / Skype API version 2.5)

Following properties were added to CHAT object in protocol 7 (API version 3.0):

- MEMBEROBJECTS – contains the list of CHATMEMBER object IDs. Refer to
  - CHATMEMBER object for list of properties
  - GET CHATMEMBER command on how to access those properties.
  - GET CHAT MEMBEROBJECTS command on how to get a list of chatmember object IDs for a given chat.

- PASSWORDHINT – contains password hint text for the chat object. Refer to ALTER CHAT SETPASSWORD command on how to set chat passwords.

GUIDELINES – contains chat guidelines text. Refer to ALTER CHAT SETGUIDELINES command on how to set chat guidelines.
- OPTIONS – bitmap of chat options. Refer to ALTER CHAT SETOPTIONS command for more information.
- DESCRIPTION – currently used only for hidden synchronization channels for managing shared groups.
- DIALOG_PARTNER – the handle of the dialog partner for dialog type chats (chats with two participants).
- ACTIVITY_TIMESTAMP – the UNIX timestamp of last activity.
- TYPE – chat type with following possible values:
  - LEGACY_DIALOG – no longer supported.
  - DIALOG – a chat with only two participants.
  - MULTICHAT – a chat with more than two participants.
  - SHAREDGROUP – a chat used for synchronization of shared contact groups.
  - LEGACY_UNSUBSCRIBED – no longer supported.

- MYSTATUS – user's current status in chat. Possible values are:
  - CONNECTING – status set when the system is trying to connect to the chat.
  - WAITING_REMOTE_ACCEPT – set when a new user joins a public chat. When the chat has "participants need authorization to read messages" option, the MYSTATUS property of a new applicant will remain in this status until he gets accepted or rejected by a chat administrator. Otherwise user's MYSTATUS will automatically change to either LISTENER or USER, depending on public chat options.
  - ACCEPT_REQUIRED – this status is used for shared contact groups functionality.
  - PASSWORD_REQUIRED – status set when the system is waiting for user to supply the chat password.
  - SUBSCRIBED – set when user joins the chat.
  - UNSUBSCRIBED – set when user leaves the chat or chat ends.
  - CHAT_DISBANDED – status set when the chat is disbanded.
  - QUEUED_BECAUSE_CHAT_IS_FULL – currently the maximum number of people in the same chat is 100.
  - APPLICATION_DENIED – set when public chat administrator has rejected user from joining.
  - KICKED – status set when the user has been kicked from the chat. Note that it is possible for the user to re-join the chat after being kicked.
  - BANNED – status set when the user has been banned from the chat.
  - RETRY_CONNECTING – status set when connect to chat failed and system retries to establish connection.

- MYROLE – user's privilege level in chat Refer to CHAT ROLES section for more information.
- BLOB – for public chats, this property contains encoded list of chat join-points. Contents of this field is used in public chat URLs.
- APPLICANTS – this property contains list of skypenames of people who have applied to join the chat but have not yet been accepted by a public chat administrator. Users only become applicants when the chat has JOINERS_BECOME_APPLICANTS option. Refer to ALTER CHAT SETOPTIONS command for more information.

CHATMEMBER object

Version
Protocol 7 (API version 3.0)

Properties:

- CHATNAME –
- IDENTITY –
- ROLE – CREATOR|MASTER|HELPER|USER|LISTENER|APPLICANT Refer to chat roles for more information.
- IS_ACTIVE – TRUE|FALSE
  - TRUE – indicates that the chat member has joined the chat.
  - FALSE indicates that the member has been added to the chat but has not yet acknowledged it. Normally occurs when the member who was added to a chat was offline at the time. Once IS_ACTIVE becomes TRUE, it will remain true.

Refer to GET CHATMEMBER command on how to access CHATMEMBER properties.

CHATMESSAGE object

Version
Protocol 3. Supersedes the MESSAGE object. Updated in protocol 7. Note that when your application connects to Skype, "PROTOCOL 7" command must be sent to Skype before your client can recognize new message types added in protocol 7. Connecting with default protocol (protocol 1) will cause new message types being reported as UNKNOWN.

Properties

- TIMESTAMP – time when message was sent (UNIX timestamp), for example MESSAGE 21 TIMESTAMP 1078958218
- PARTNER_HANDLE – **NB!** This property is deprecated since API version 3.0 and replaced with FROM_HANDLE.
- PARTNER_DISPNAME – **NB!** This property is deprecated since API version 3.0 and replaced with FROM_DISPNAME.
- FROM_HANDLE – skypename of the originator of the chatmessage.
- FROM_DISPNAME – displayed name of the originator of the chatmessage.
- TYPE – message type, for example MESSAGE 21 TYPE TEXT . Possible values:
  - SETTOPIC – change of chat topic
  - SAID – IM
  - ADDEDMEMBERS – invited someone to chat
  - SAWMEMBERS – chat participant has seen other members
  - CREATEDCHATWITH – chat to multiple people is created
  - LEFT – someone left chat; can also be a notification if somebody cannot be added to chat
  - POSTEDCONTACTS – system message that is sent or received when one user sends contacts to another. Added in protocol 7.
  - GAP_IN_CHAT – messages of this type are generated locally, during synchronization, when a user enters a chat and it becomes apparent that it is impossible to update user's chat history completely. Chat history is kept only up to maximum of 400 messages or 2 weeks. When a user has been offline past that limit, GAP_IN_CHAT notification is generated. Added in protocol 7.
  - SETROLE – system messages that are sent when a chat member gets promoted or demoted. Refer to ALTER CHATMEMBER SETROLETO command for more info on how to change chat member roles. Added in protocol 7.
  - KICKED – system messages that are sent when a chat member gets kicked. Refer to ALTER CHAT KICK command for more information. Added in protocol 7.
  - KICKBANNED – system messages that are sent when a chat member gets banned. Refer to ALTER CHAT KICKBAN command for more information. Added in protocol 7.
  - SETOPTIONS – system messages that are sent when chat options are changed. Refer to ALTER CHAT SETOPTIONS command for more information. Added in protocol 7.
  - SETPICTURE – system messages that are sent when a chat member has changed the public chat topic picture. Added in protocol 7.
  - SETGUIDELINES – system messages that are sent when chat guidelines are changed. Refer to ALTER CHAT SETGUIDELINES command for more information. Added in protocol 7.
  - JOINEDASAPPLICANT – notification message that gets sent in a public chat with JOINERS_BECOME_APPLICANTS options, when a new user joins the chat. See ALTER CHAT SETOPTIONS command for more information on chat options. Added in protocol 7.
  - UNKNOWN – unknown message type, possibly due to connecting to Skype with older protocol.

- STATUS – message status, for example MESSAGE 21 STATUS QUEUED . Possible 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, for example CHATMESSAGE 21 LEAVEREASON UNSUBSCRIBE . Possible values:
  - USER_NOT_FOUND – user was not found
  - USER_INCAPABLE – user has an 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

- CHATNAME – chat that includes the message, for example #test_3/$b17eb511457e9d20
- USERS – people added to chat
- IS_EDITABLE – TRUE|FALSE Refer to SET CHATMESSAGE BODY command for more information on how to edit chat message text (BODY) and on what conditions is such editing permitted. This property was introduced in API version 3.0
- EDITED_BY – identity of the last user who edited the message. New in API version 3.0
- EDITED_TIMESTAMP – UNIX timestamp of the last edit. New in API version 3.0
- OPTIONS – numeric field that contains chat options bitmap in system messages that get sent out when a change is made to chat options (messages where TYPE is SETOPTIONS). In normal messages the value of this field is 0. Refer to ALTER CHAT SETOPTIONS command for more information.
- ROLE – used in system messages that get sent when a public chat administrator has promoted or demoted a chat member. The TYPE property of such messages is set to SETROLE. In these messages the value of this field is set to the new assigned role of the promoted or demoted chat member. In normal messages the value of this property is set to UNKNOWN. Refer to CHAT ROLES section for a list of different chat roles and ALTER CHATMEMBER SETROLETO command for how chat roles can be changed. New in API version 3.0

Most CHATMESSAGE properties are read-only. The following property is read-write and can be modified with the SET command:

- SEEN – mark missed chatmessage as seen and removes chat from missed events.
- BODY – message text. Note that this property was read-only prior to API version 3.0

VOICEMAIL object

Version
Protocol 5

Properties

- TYPE – type of voicemail object
  - INCOMING – voicemail received from partner
  - OUTGOING – voicemail sent to partner
  - DEFAULT_GREETING – Skype default greeting from partner
  - CUSTOM_GREETING – partner's recorded custom greeting
  - UNKNOWN
- PARTNER_HANDLE – username for voicemail sender (for incoming) or recipient (for outgoing)
- PARTNER_DISPNAME – user displayname for partner
- STATUS – current status of voicemail object
  - NOTDOWNLOADED – voicemail is stored on server (has not been downloaded yet)
  - DOWNLOADING – downloading from server to local machine
  - UNPLAYED – voicemail has been downloaded but not played back yet
  - BUFFERING – buffering for playback
  - PLAYING – currently played back
  - PLAYED – voicemail has been played back
  - BLANK – intermediate status when new object is created but recording has not begun
  - RECORDING – voicemail currently being recorded
  - RECORDED – voicemail recorded but not yet uploaded to the server
  - UPLOADING – voicemail object is currently being uploaded to server
  - UPLOADED – upload to server finished but not yet deleted; object is also locally stored
  - DELETING – pending delete
  - FAILED – downloading voicemail/greeting failed
  - UNKNOWN
- FAILUREREASON possible values
  - MISC_ERROR
  - CONNECT_ERROR
  - NO_VOICEMAIL_PRIVILEGE
  - NO_SUCH_VOICEMAIL
  - FILE_READ_ERROR
  - FILE_WRITE_ERROR
  - RECORDING_ERROR
  - PLAYBACK_ERROR
  - UNKNOWN
- SUBJECT – not used
- TIMESTAMP
- DURATION – actual voicemail duration in seconds
- ALLOWED_DURATION – maximum voicemail duration in seconds allowed to leave to partner
- INPUT – New in API version 3.5.0.202 Can have following values:
  - SOUNDCARD="default" – default is currently the only acceptable value.
  - PORT="port_no" – the ID of the audio port (1..65535)
  - FILE="filename.wav" – the path and name of the audio file.
- OUTPUT – can have all the same values as INPUT property. New in API version 3.5.0.202
- CAPTURE_MIC – can have all the same values as INPUT and OUTPUT properties. New in API version 3.5.0.202

SMS object

Version
Added in API version 2.5

Refer to Sending and managing SMS messages section for additional info.

Properties

- BODY – SMS message text, read-write access
- TYPE – Possible values:
  - INCOMING – received messages. Note that as sending SMS messages to Skype numbers is currently not supported, this status is here mainly for future compatibility.
  - OUTGOING – sent messages
  - CONFIRMATION_CODE_REQUEST – [#SMS_NUMBER_VALIDATION used for registering user's Skype ID as a reply-to number]
  - CONFIRMATION_CODE_SUBMIT – [#SMS_NUMBER_VALIDATION used for registering user's Skype ID as a reply-to number]
  - UNKNOWN – for unknown reasons, the message type is unknown

- STATUS – Possible values:
  - RECEIVED – the message has been received (but not tagged as read)
  - READ – the message has been tagged as read
  - COMPOSING – the message has been created but not yet sent
  - SENDING_TO_SERVER – the message is in process of being sent to server
  - SENT_TO_SERVER – the message has been sent to server
  - DELIVERED – server has confirmed that the message is sent out to recepient
  - SOME_TARGETS_FAILED – server reports failure to deliver the message to one of the recepients within 24h
  - FAILED – the message has failed, possible reason may be found in FAILUREREASON property
  - UNKNOWN – message status is unknown

- FAILUREREASON
  - MISC_ERROR – indicates failure to supply a meaningful error message
  - SERVER_CONNECT_FAILED – unable to connect to SMS server
  - NO_SMS_CAPABILITY – recepient is unable to receive SMS messages
  - INSUFFICIENT_FUNDS – insufficient Skype Credit to send an SMS message
  - INVALID_CONFIRMATION_CODE – set when an erroneous code was submitted in a CONFIRMATION_CODE_SUBMIT message
  - USER_BLOCKED – user is blocked from the server
  - IP_BLOCKED – user's IP is blocked from the server
  - NODE_BLOCKED – user's p2p network node has been blocked from the server
  - UNKNOWN – default failure code
  - NO_SENDERID_CAPABILITY – Set when a CONFIRMATION_CODE_REQUEST SMS message is sent with a mobile phone number containing country code of either USA, Taiwan or China. Setting reply-to number from Skype SMS's to your mobile number is not supported in these countries. Added in Skype version 3.5 (protocol 8).

- IS_FAILED_UNSEEN – TRUE|FALSE To change this value from True to False, use SET SMS <id> SEEN command
- TIMESTAMP – Unix timestamp (usually GMT)
- PRICE – cost of sending the SMS message (integer value)
- PRICE_PRECISION – 1|2|3.. the number of times the PRICE is divided by 10 to express ther full currency unit.
  For example, a PRICE of 1234 with PRICE_PRECISION of 2 amounts to 12.34.
- PRICE_CURRENCY – EUR|USD..
- REPLY_TO_NUMBER – reply-to field of the SMS message, read-write access
- TARGET_NUMBERS – comma-separated list of SMS recepients (+number, +number, +number..), read-write access
- TARGET_STATUSES – a string containing comma-separated list of recepients with message delivery status for each of them, in following format: "+number=status, +number=status.." Possible values for target statuses are:
  - TARGET_ANALYZING
  - TARGET_UNDEFINED
  - TARGET_ACCEPTABLE
  - TARGET_NOT_ROUTABLE
  - TARGET_DELIVERY_PENDING
  - TARGET_DELIVERY_SUCCESSFUL
  - TARGET_DELIVERY_FAILED
  - UNKNOWN

APPLICATION object

Properties

CONNECTABLE – query connectable users. NB! From API version 3.0, this property enters the deprecation process.

```
-> GET APPLICATION appname CONNECTABLE
<- APPLICATION appname CONNECTABLE [username[ username]*]
```

CONNECTING – query on-going connection process after the connection is established. Username is removed from CONNECTING list.

```
-> GET APPLICATION appname CONNECTING
<- APPLICATION appname CONNECTING [username[ username]*]
```

STREAMS – query open streams (connections)

```
-> GET APPLICATION appname STREAMS
<- APPLICATION appname STREAMS [username:id[ username:id]*]
```

SENDING – query if currently sending any data. After the data is sent, the stream name is removed from the SENDING list

```
-> GET APPLICATION appname RECIEVED
<- APPLICATION appname SENDING [username:id=bytes [username:id bytes]*]
```

Note: In Skype 1.4x, the number of bytes reported by the SENDING notification following an APPLICATION WRITE is 2 bytes longer than that which was written.

```
-> alter application exe write testtest20:1 w
<- ALTER APPLICATION exe WRITE testtest20:1
<- APPLICATION exe SENDING testtest20:1 3
-> alter application exe write testtest20:1 1234567890
<- ALTER APPLICATION exe WRITE testtest20:1
<- APPLICATION exe SENDING testtest20:1 12
```

RECEIVED – query if there is data waiting in received buffer. After the data is read from the stream, the stream name is removed from the RECEIVED list.

```
-> GET APPLICATION appname RECEIVED
<- APPLICATION appname SENDING [username:id=bytes [username:id bytes]*]
```

incoming datagram notification

```
<- APPLICATION appname DATAGRAM user:id text
```

Version

- Protocol 5
- Skype for Windows 1.4

GROUP object

The GROUP object enables users to group contacts. There are two types of GROUP ; custom groups and hardwired groups. The GROUP object has the following properties:

- TYPE: {ALL | CUSTOM | HARDWIRED | SHARED_GROUP | PROPOSED_SHARED_GROUP}
  - ALL – all groups. (new in API version 2.5)
  - CUSTOM – user-defined groups.
  - HARDWIRED – "smart" groups defined by Skype to manage groups.
  - SHARED_GROUP – shared groups, with semi-automatic cross-authrization between contacts (new in API version 2.5)
  - PROPOSED_SHARED_GROUP – a group that has turned into a shared group and is waiting for accept/decline (new in API version 2.5)
- CUSTOM_GROUP_ID – a persistent ID for custom groups which can be empty at the start of group creation.
- DISPLAYNAME – the display name of the group (read-write)

- NROFUSERS – the number of contacts in this group (read-only)
- NROFUSERS_ONLINE – the number of contacts online in this group (read-only)
- USERS – the list of contacts in the group (read-only)
  Following is a description of all group types defined by Skype:

HARDWIRED GROUPS are described in the following table.

Contact group type	Description
ALL_USERS	This group contains all users I know about, including users in my contactlist, users I recently contacted and blocked users.
ALL_FRIENDS	This group contains all contacts in my contactlist (also known as friends)
SKYPE_FRIENDS	This group contains Skype contacts in my contactlist.
SkypeOut_FRIENDS	This group contains SkypeOut contacts in my contactlist.
ONLINE_FRIENDS	This group contains Skype contacts in my contactlist who are online.
UNKNOWN_OR_PENDINGAUTH_FRIENDS	This group contains contacts in my contactlist who have not yet authorized me.
RECENTLY_CONTACTED_USERS	This group contains contacts I have conversed with recently, including non-friends.
USERS_WAITING_MY_AUTHORIZATION	This group contains contacts who are awating my response to an authorisation request, including non-friends.
USERS_AUTHORIZED_BY_ME	This group contains all contacts I have authorised, including non-friends.
USERS_BLOCKED_BY_ME	This group contains all contacts I have blocked, including non-friends.
UNGROUPED_FRIENDS	This group contains all contacts in my contactlist that do not belong to any custom group.
CUSTOM_GROUP	This group type is reserved for user-defined groups.

FILETRANSFER object

File transfer objects are for monitoring purposes only. No alters/actions via API are currently allowed with these objects. File transfers cannot be initiated nor accepted via API commands.

Values of all the properties can be accessed with `GET FILETRANSFER <id> <property_name>` commands.

Refer to SEARCH FILETRANSFERS and SEARCH ACTIVEFILETRANSFERS for getting lists of FILETRANSFER objects in the system.

Properties:

- TYPE – possible values are:
  - INCOMING – file transfer object from receiving side.
  - OUTGOING – file transfer object from transmitting side.
- STATUS – current status of the object. Possible values are:
  - NEW – initial state of a file transfer. For sender, the status proceeds to WAITING_FOR_ACCEPT.
  - WAITING_FOR_ACCEPT – status set for sender until receiver either accepts or cancels the transfer.
  - CONNECTING – is set for both parties after remote user accepts the file transfer.
  - TRANSFERRING – is set at the start of the file transfer.
  - TRANSFERRING_OVER_RELAY – set when no direct connection between sender and receiver could be established over the network. Analogous to TRANSFERRING.

    - PAUSED – this status is currently unused.
    - REMOTELY_PAUSED – this status is also currently unused.
    - CANCELLED – file transfer has been locally cancelled. Remote user status is set to FAILED and FAILURE_REASON to REMOTELY_CANCELLED.
    - COMPLETED – file transfer was completed.
    - FAILED – file transfer failed to complete. Cause of the failure can be seen in FAILUREREASON.

- FAILUREREASON – set when STATUS is set to FAILED.
    - SENDER_NOT_AUTHORIZED – It is only possible to transfer files between users who have authorized each-other. As initiating file transfers to remote users who have not authorized the sender is currently blocked by UI, this FAILUREREASON appears to be unused.
    - REMOTELY_CANCELLED – set when remote user has cancelled the transfer.
    - FAILED_READ – read error on local machine.
    - FAILED_REMOTE_READ – read error on remote machine.
    - FAILED_WRITE – write error on local machine.
    - FAILED_REMOTE_WRITE – write error on remote machine.
    - REMOTE_DOES_NOT_SUPPORT_FT – Skype client of the receiver does not support file transfers.
    - REMOTE_OFFLINE_FOR_TOO_LONG – the recipient of the proposed file transfer is not available (offline for longer than 7 days).

- PARTNER_HANDLE – remote user's skypename.
- PARTNER_DISPNAME – remote user's display name.
- STARTTIME – Unix timestamp of when the transfer was started.
- FINISHTIME – while transmission is in progress the value is updated with estimated time of completion (0 when no estimation can be given). When transmission is finished, the value is set to the timestamp of completion/failure.
- FILEPATH – full path of the file being read or written in local file system. Includes filename and extension. * FILENAME – filename (and extension) without path. This is also seen by the receiver before accept (default file name, from sender).
- FILESIZE – file size, 64-bit numeric.
- BYTESPERSECOND – transfer speed during file transfer. Becomes 0 after transfer is completed, failed or aborted.
- BYTESTRANSFERRED – current nr. of bytes transferred (progress), 64-bit numeric.

Example:

```
//-------------------------------------------------------------------------------
// Sender initiates file transfer from UI
// Note that the file name in notification message is not enclosed in quotes.
<- FILETRANSFER 982 TYPE OUTGOING
<- FILETRANSFER 982 PARTNER_HANDLE Test2
<- FILETRANSFER 982 PARTNER_DISPNAME Test2
<- FILETRANSFER 982 FILEPATH C:\Stuff\This is test file.mp3
<- FILETRANSFER 982 FILENAME This is test file.mp3
<- FILETRANSFER 982 STATUS NEW
<- FILETRANSFER 982 FILESIZE 0
<- FILETRANSFER 982 STARTTIME 1174558044
<- FILETRANSFER 982 FINISHTIME 0
<- FILETRANSFER 982 BYTESPERSECOND 0
<- FILETRANSFER 982 BYTESTRANSFERRED 0
<- FILETRANSFER 982 FILESIZE 2193720
<- FILETRANSFER 982 STATUS WAITING_FOR_ACCEPT
//-------------------------------------------------------------------------------
// Remote user receives incoming file notification
<- FILETRANSFER 1250 TYPE INCOMING
<- FILETRANSFER 1250 PARTNER_HANDLE Test
<- FILETRANSFER 1250 PARTNER_DISPNAME Test
<- FILETRANSFER 1250 FILENAME This is test file.mp3
<- FILETRANSFER 1250 STATUS NEW
<- FILETRANSFER 1250 STARTTIME 1174644373
<- FILETRANSFER 1250 FINISHTIME 0
<- FILETRANSFER 1250 BYTESPERSECOND 0
<- FILETRANSFER 1250 BYTESTRANSFERRED 0
//-------------------------------------------------------------------------------
// Remote user accepts the file from UI and starts receiving
<- FILETRANSFER 1250 FILEPATH C:\test\This is test file.mp3
<- FILETRANSFER 1250 STATUS CONNECTING
<- FILETRANSFER 1250 STATUS TRANSFERRING
<- FILETRANSFER 1250 BYTESTRANSFERRED 262454
<- FILETRANSFER 1250 BYTESPERSECOND 307806
<- FILETRANSFER 1250 BYTESTRANSFERRED 580110
<- FILETRANSFER 1250 FINISHTIME 1174644526
<- FILETRANSFER 1250 BYTESPERSECOND 526959
<- FILETRANSFER 1250 BYTESTRANSFERRED 1316372
```

```
<- FILETRANSFER 1250 BYTESPERSECOND 613776
<- FILETRANSFER 1250 BYTESTRANSFERRED 2103782
<- FILETRANSFER 1250 BYTESPERSECOND 0
<- FILETRANSFER 1250 BYTESTRANSFERRED 2193720
<- FILETRANSFER 1250 STATUS COMPLETED
//-------------------------------------------------------------------------------
// Sender receives notification that the file has been accepted and starts sending
<- FILETRANSFER 982 STATUS CONNECTING
<- FILETRANSFER 982 STATUS TRANSFERRING
<- FILETRANSFER 982 BYTESTRANSFERRED 262454
<- FILETRANSFER 982 BYTESPERSECOND 308104
<- FILETRANSFER 982 BYTESTRANSFERRED 580110
<- FILETRANSFER 982 FINISHTIME 1174558198
<- FILETRANSFER 982 BYTESPERSECOND 510987
<- FILETRANSFER 982 BYTESTRANSFERRED 1296182
<- FILETRANSFER 982 FINISHTIME 1174558195
<- FILETRANSFER 982 BYTESPERSECOND 606237
<- FILETRANSFER 982 BYTESTRANSFERRED 2083592
<- FILETRANSFER 982 BYTESPERSECOND 0
<- FILETRANSFER 982 FINISHTIME 0
<- FILETRANSFER 982 STATUS CONNECTING
<- FILETRANSFER 982 BYTESTRANSFERRED 2193720
<- FILETRANSFER 982 FINISHTIME 1174558195
<- FILETRANSFER 982 STATUS COMPLETED
```

Version

Protocol 7 (API version 3.0)

Managing object properties

Three commands are available for retrieving and modifing object properties and general parameters:

- GET – general request command to retrieve object properties and general parameters
- SET – to set object properties and modify general parameters
- ALTER – to alter or perform an action with an object

General syntax

- `GET USER <username> property`
  `|CALL <id> property`
  `|MESSAGE <id> property`
  `|CHAT <id> property`
  `|CHATMESSAGE <id> property`
  `|VOICEMAIL <id> property`
- `SET USER <username> property <value>`
  `|CALL <id> property <value>`
  `|MESSAGE <id> property <value>`
  `|CHAT <id> property <value>`
  `|CHATMESSAGE <id> property <value>`
  `|VOICEMAIL <id> property <value>`
  See the corresponding object information for available properties and property values:
- CALL object
- USER object
- PROFILE object
- CHAT object
- CHATMESSAGE object
- VOICEMAIL object
- APPLICATION object
  This section contains the commands for managing object properties. Note that:
- The `GET MESSAGE` command is deprecated and has been replaced by the GET CHATMESSAGE command.
- Commands for the `APPLICATION` object are described in the APPLICATION object information.

GET USER

This command returns property values for a specified user.

Syntax
`GET USER <username> property`

Response
`USER <username> property <value>`

Parameters

- `<username>` – Skype username to retrieve property
- `property` – property name. Refer to the USER object information for list of properties.

Version
Protocol 1

Errors

- `ERROR 7 GET: invalid WHAT`
  Object name missing or misspelled
- `ERROR 10 Invalid prop`
  ID and/or property missing or misspelled.
- `ERROR 8 invalid handle`
  `USERNAME` missing or includes a not permitted character . Note: The `GET USER <target> ONLINESTATUS` command returns the response `OFFLINE` unless the current user is authorized by the target user to see his/her online status.

Example

```
-> GET USER pamela FULLNAME
<- USER pamela FULLNAME Jane Doe
```

SET USER

Syntax
`SET USER <target> ISAUTHORIZED TRUE|FALSE` – allow/disable target to see current user's userstatus
`SET USER <target> ISBLOCKED TRUE|FALSE` – block/unblock target user
`SET USER <target> BUDDYSTATUS 1` – remove target from contactlist
`SET USER <target> BUDDYSTATUS 2 <message>` – add target into contactlist and ask authorization with message

GET CALL

This command returns property values for a specified call. See GET CALL command reference for more details.

GET CHAT

This command returns property values for a specified chat.

Syntax
`GET CHAT <chat_id> property`

Response
`CHAT <chat_id> property <value>`

Parameters

- `<chat_id>` – chat identifier;
- `property` – property name.
  Available properties are: `NAME` , `TIMESTAMP` , `ADDER` , `STATUS` , `POSTERS` , `MEMBERS` , `TOPIC` , `CHATMESSAGES` , `ACTIVEMEMBERS` , `FRIENDLYNAME` . See CHAT object description for detailed info.

Protocol 3

Errors

- `ERROR 7 GET: invalid WHAT`
  Object name missing or misspelled.
- `ERROR 105 invalid chat name`
  Errorin the `CHATNAME` parameter.
- `ERROR 106 Invalid PROP`
  Property name missing or misspelled.

Example

```
-> GET CHAT #bitman/$jessy;eb06e65635359671 NAME
<- CHAT #bitman/$jessy;eb06e65635359671 NAME #bitman/$jessy;eb06e65635359671
```

GET CHATMESSAGE

This command returns property values for a specified chat message.

Syntax
`GET CHATMESSAGE <id> property`

Response
`CHATMESSAGE <id> property <value>`

Parameters

- `<id>` – chat message ID;
- `property` – property name.
  Available properties are: `CHATNAME` , `TIMESTAMP` , `FROM_HANDLE` , `FROM_DISPNAME` , `TYPE` , `USERS` , `LEAVEREASON` , `BODY` , `STATUS` . Refer to the CHATMESSAGE object information for more detail.

Version
Protocol 3

Example

```
-> GET CHATMESSAGE 60 CHATNAME
<- CHATMESSAGE 60 CHATNAME #bitman/$jessy;eb06e65631239671
```

Errors

- `ERROR 7 GET: invalid WHAT`
  Object name missing or misspelled.
- `ERROR 14 Invalid message id`
  Chat message ID contains not permitted symbols (only numeric are permitted)
- `ERROR 15 Unknown message`
  Unknown chat message ID
- `ERROR 16 Invalid PROP`
  Property name missing or misspelled

GET MESSAGE

This command returns property values for a specified message. This command is deprecated since protocol 3, and was replaced by the GET CHATMESSAGE command.

Syntax
`GET MESSAGE <id> property`

Parameters

- `<id>` – chat message ID;

property - property name.
Availableproperties are: `TIMESTAMP (UNIX timestamp)`, `PARTNER_HANDLE`, `PARTNER_DISPNAME`, `CONF_ID (not used)`, `TYPE`, `STATUS`, `FAILUREREASON (numeric)`, `BODY`. Refer to the MESSAGE object information for more detail.

Version
Protocol 1, **deprecated in protocol 3**

Errors

- `ERROR 7 GET: invalid WHAT`
  Object name missing or misspelled.
- `ERROR 14 Invalid message id`
  ID includes other than numeric characters.
- `ERROR 15 Unknown message`
  Message with specified ID does not exist in current user's message history.
- `ERROR 16 Invalid prop`
  Property name missing or misspelled.

Example

```
-> GET MESSAGE 159 TYPE
<- MESSAGE 159 TYPE TEXT
```

GET APPLICATION

For information about the `GET APPLICATION` command, refer to the APPLICATION object information.

Managing general parameters

Use `GET` and `SET` commands to manage the general variables.

GET SKYPEVERSION

Syntax
`GET SKYPEVERSION`

Response
`SKYPEVERSION <version>`

Version
Protocol 1

Example

```
-> GET SKYPEVERSION
<- SKYPEVERSION 1.3.0.28
```

GET CURRENT USER

This command gets the username for the currently logged in user.

Syntax
`GET CURRENTUSERHANDLE`

Response
`CURRENTUSERHANDLE <username>`

Version
Protocol 1

GET USERSTATUS

This command queries or modifies user visiblity for the current user.

Syntax
```
GET USERSTATUS
SET USERSTATUS <value>
```

Response
```
USERSTATUS <value>
```

Parameters
`<value>` – new userstatus. Possible 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.

Version
Protocol 1

Errors

- `ERROR 28 Unknown userstatus`
  Status value is incorrect or misspelled

Example

```
-> SET USERSTATUS OFFLINE
<- USERSTATUS OFFLINE
<- USERSTATUS OFFLINE
-> SET USERSTATUS xxx
<- ERROR 28 Unknown userstatus
```

GET PRIVILEGE

Syntax
```
GET PRIVILEGE user_privilege
```

Response
```
PRIVILEGE user_privilege <value>
```

Parameters

- `user_privilege` – possible values:
  - `SkypeOut True or False`
  - `SkypeIn True or False`
  - `VOICEMAIL True or False`

Errors

- `ERROR 40 Unknown privilege`
  Privilege name is missing or misspelled

Version
Protocol 1

Example

```
-> GET PRIVILEGE SkypeOut
<- PRIVILEGE SkypeOut TRUE
-> GET PRIVILEGE SkypeIn
<- PRIVILEGE SkypeIn FALSE
```

GET PROFILE

This command queries the current user's profile information.

Syntax:

```
-> GET PROFILE <profile_property>

<- PROFILE <profile_property> <value>
```

Refer to PROFILE object for possible values of parameter.

Example:

```
-> GET PROFILE PSTN_BALANCE
<- PROFILE PSTN_BALANCE 5000
-> GET PROFILE PSTN_BALANCE_CURRENCY
<- PROFILE PSTN_BALANCE_CURRENCY EUR
```

Version
Protocol 3

GET PREDICTIVE DIALER COUNTRY

This command returns the country code that is currently being used for inventing correct country prefixes for PSTN numbers (predictive dialing). The country code is returned in ISO2 format.

Syntax:

```
-> GET PREDICTIVE_DIALER_COUNTRY

<- PREDICTIVE_DIALER_COUNTRY <iso2>
```

Example:

```
-> GET PREDICTIVE_DIALER_COUNTRY
<- PREDICTIVE_DIALER_COUNTRY ee
```

Version
Protocol 7 (API version 3.1)

SET PROFILE MOOD_TEXT

The `SET PROFILE MOOD TEXT` command changes the mood text for a user.

Syntax

```
-> SET PROFILE MOOD_TEXT Life is great and then you...

<- PROFILE MOOD_TEXT Life is great and then you...
```

Version
Protocol 5

SET PROFILE RICH_MOOD_TEXT

This is a "with bells and whistles" version of the `SET PROFILE MOOD_TEXT` command.

Syntax:

```
-> SET PROFILE RICH_MOOD_TEXT <text>

<- PROFILE RICH_MOOD_TEXT <text>

<- PROFILE RICH_MOOD_TEXT <text>

<- PROFILE MOOD_TEXT <text>

<- USER <username> RICH_MOOD_TEXT <text>

<- USER <username> MOOD_TEXT <text>
```

Note that when this property is changed, it is also propagated into the old `MOOD_TEXT`, with XML tags stripped. Corresponding properties of the `USER` object are updated as well.

When `MOOD_TEXT` property is set, the `RICH_MOOD_TEXT` property is automatically cleared.

Example:

```
//---------------------------------------------------------------
// For purpose of bit conservation we omit feedback notifications
SET PROFILE RICH_MOOD_TEXT Smiley: <SS type="smile">:-)</SS>
SET PROFILE RICH_MOOD_TEXT <FONT COLOR="#FF0010">Red text</FONT>
SET PROFILE RICH_MOOD_TEXT <BLINK>Blinking text</BLINK>
SET PROFILE RICH_MOOD_TEXT <B>Bold text</B>
SET PROFILE RICH_MOOD_TEXT <I>Italics</I>
SET PROFILE RICH_MOOD_TEXT <U>Underlined</U>
SET PROFILE RICH_MOOD_TEXT First line<br/>Second line<br/>Third line
```

`<SS type="smile"></SS>` also accepts following smileys:

smile, sad, laugh, cool, surprised, wink, cry, sweat, speechless, kiss, tongueout, blush, wonder, sleepy, snooze, dull, inlove, talk, yawn, puke, doh, angry, wasntme, party, worry, mmm, nerdy, lipssealed, hi, call, devil, angel, envy, wait, hug, makeup, giggle, clap, think, bow, rofl, whew, happy, smirk, nod, shake, punch, emo, no, yes, handshake, skype, heart, brokenheart, mail, flower, rain, sun, time, music, movie, phone, coffee, pizza, cash, muscle, beer, drink, dance, ninja, star, mooning, finger, bandit, smoke, toivo, rock, headbang, poolparty, swear, bug, fubar, tmi.

You can also get ideas for cute mood messages by looking at what others have done with theirs. To retrieve rich mood messages of other people, use GET USER RICH MOOD TEXT command.

Version
Protocol 7 (API version 3.0)

GET USER RICH_MOOD_TEXT

Retrieves RICH_MOOD_TEXT of a remote user.

Syntax:

```
-> GET USER <skypename> RICH_MOOD_TEXT

<- USER <skypename> RICH_MOOD_TEXT <text>
```

Version
Protocol 7 (API version 3.0)

GET CONNSTATUS (connection)

This command returns the current network connection status.

Syntax
`GET CONNSTATUS`

Response
`CONNSTATUS <value>`

Parameters
`<value>` – possible values:

- `OFFLINE`
- `CONNECTING`
- `PAUSING`
- `ONLINE`

Version
Protocol 1

Example

```
-> GET CONNSTATUS
<- CONNSTATUS ONLINE
```

AUDIO_IN

The GET command returns the current audio input device for Skype.
The SET command assigns a new audio input device for Skype.

Syntax
`GET AUDIO_IN`
`SET AUDIO_IN <device_name>`

Response
`AUDIO_IN <device_name>`

Version
Protocol 1

Note
Setting a device with an empty name selects the Windows default device.

Example

```
-> GET AUDIO_IN
<- AUDIO_IN SB Audigy 2 ZS Audio [DC00]
```

AUDIO_OUT

The GET command returns the current audio output device for Skype.
The SET command assigns a new audio output device for Skype.

Syntax
`GET AUDIO_OUT`
`SET AUDIO_OUT <device_name>`

Response
AUDIO_OUT <device_name>

Version
Protocol 1

Note
Setting a device with an empty name selects the Windows default device.

Example

```
-> GET AUDIO_OUT
<- AUDIO_OUT SB Audigy 2 ZS Audio [DC00]
```

RINGER

The GET command returns the current ringing device for Skype. The SET command assigns a new ringing device for Skype.

Syntax
GET RINGER
SET RINGER <device_name>

Response
RINGER <device_name>

Version
Skype for Windows 1.3

Note
Setting a device with an empty name selects the Windows default device.

Example

```
-> GET RINGER
<- RINGER SB Audigy 2 ZS Audio [DC00]
```

MUTE

This command gets or sets the mute status.

Syntax
GET MUTE
SET MUTE ON|OFF

Response
MUTE ON|OFF

Version
Protocol 1

Notes
If there are currently no active calls (call status INPROGRESS), MUTE is always OFF and setting MUTE ON has no effect.

Example

```
-> GET MUTE
<- MUTE OFF
// set mute when no call is active - mute remains OFF
-> SET MUTE ON
<- MUTE OFF
```

SET AVATAR

This command changes the avatar picture for the user profile.

Syntax
```
SET AVATAR <id> <filePath + fileName>[:idx]
```

Response
```
AVATAR <id> <filePath + fileName>
```

Parameters

- `id` – avatar ID. This parameter is here for future compatibility purposes. Currently only one avatar is supported, so always set this parameter to '1'.
- `filePath` – avatar file directory.
- `fileName:idx` – avatar file may either be image or .skype file format. IDX refers to the content number in .skype file formats (0,..)

Version

- Skype for Windows 1.3
- .skype files are supported in Skype for Windows 1.4
- Protocol 5 supports changing avatars.

Errors

- `ERROR 114 Invalid avatar`
  Avatar id is missing or invalid
- `ERROR 111 File not found`
  Avatar file specified does not exist
- `ERROR 9901 internal error`
  Wrong type of file (for example an audio file or a document) is set to avatar

Example

```
-> SET AVATAR 1 C:\Documents and Settings\Administrator\My Documents\My Pictures\kitten.jpg
<- AVATAR 1 C:\Documents and Settings\Administrator\My Documents\My Pictures\kitten.jpg
```

GET AVATAR

This command saves user's current avatar picture in a file.

Refer to

- GET USER AVATAR command for how to save avatars of other users.
- SET AVATAR command on how to set your own avatar to a picture from a file.

Syntax:
```
-> GET AVATAR 1 <filename>

<- AVATAR 1 <filename>
```

The file path given in the parameter must exist. An existing file with the same name will only be overwritten if it's empty (file size = 0).

Example:

```
-> GET AVATAR 1 c:\stuff\test2.jpg
<- AVATAR 1 c:\stuff\test2.jpg
```

Version
Protocol 7 (API version 3.1)

GET USER AVATAR

This command retrieves remote user's avatar picture from the picture cache and saves it into a file. Refer to SET AVATAR command on how to set your own avatar to a picture from a file.

Syntax:

```
-> GET USER <skypename> AVATAR 1 <filename>

<- USER <skypename> AVATAR 1 <filename>
```

The file path given in the parameter must exist. An existing file with the same name will only be overwritten if it's empty (file size = 0).

Example:

```
-> GET USER anappo2 AVATAR 1 c:\stuff\userpic.jpg
<- USER anappo2 AVATAR 1 c:\stuff\userpic.jpg
```

Version
Protocol 7 (API version 3.1)

RINGTONE

The GET command returns the current ringtone file for Skype.
The SET command assigns a new ringtone for Skype.

Syntax

- `GET RINGTONE <id>`
- `SET RINGTONE <id> <filePath + fileName>[:idx]`

Response

```
RINGTONE <id> <filePath + fileName>
```

Parameters

- `id` – ringtone id. In the current release, the is always '1'
- `filePath` – ringtone file directory.
- `fileName:idx` – ringtone file may either be .wav or .skype file format. IDX refers to the content number in .skype file formats (0,..)

Version

- Skype for Windows 1.3
- .skype files are supported since Skype for Windows 1.4
- Querying ringtone status is supported since Skype for Windows 1.4

Errors

- `ERROR 115 Invalid ringtone`
  Ringtone id is missing or invalid
- `ERROR 111 File not found`
  Ringtone file specified does not exist

Notes

- If the Skype default ringtone is used, the GET command returns its name with no filepath.
- .skype may be used instead of .wav files and can contain multiple contents enumerated by integer IDs (idx).

Example

```
-> GET RINGTONE 1
<- RINGTONE 1 call_in
```

```
<- RINGTONE 1 C:/WINDOWS/Media/tada.wav
```

GET RINGTONE STATUS

This command queries if ringtones are enabled.

Syntax

```
-> GET RINGTONE <id> STATUS

<- RINGTONE <id> <ON|OFF>
```

Note that the parameter is there for possible future use and must for now be always set to 1.

SET RINGTONE STATUS

This command enables you to switch ringtone ON/OFF.

Syntax:

*@→ SET RINGTONE STATUS ON|OFF@

*@<- RINGTONE ON|OFF@

Example:

```
-> SET RINGTONE 1 STATUS OFF
<- RINGTONE 1 OFF
-> GET RINGTONE 1 STATUS
<- RINGTONE 1 OFF
-> SET RINGTONE 1 STATUS ON
<- RINGTONE 1 ON
-> GET RINGTONE 1 STATUS
<- RINGTONE 1 ON
```

Note that the parameter is there for possible future use and must for now be always set to 1.

Version

Protocol 7 (API version 3.1)

GET VIDEO_IN

This command queries or sets the device to be used in video calls. See GET VIDEO_IN command reference for more details.

SET PCSPEAKER

If no speakers are connected to a PC, it is possible to hear incoming Skype calls only when wearing a headset. Use the SET PCSPEAKER command to switch the PC speaker on or off.

Syntax

```
-> GET PCSPEAKER
-> SET PCSPEAKER {ON|OFF}
```

Response

```
<- PCSPEAKER {ON|OFF}
```

SET AGC and SET AEC

NB! As of version 3.6 these commands no longer actually function. The API commands are still valid, for backward compatibility reasons, but turning echo cancellation or microphone gain off programmatically is disabled in the library.

Skype uses automatic gain control (AGC) to adjust microphone level to the volume the user speaks at. Skype uses automatic echo cancellation (AEC) to eliminate the echo that occurs if a microphone "hears" the other user's voice on the loudspeaker.

Important: Disabling these functions can impair call quality and is not recommended in standard implementations. However, some audio devices have in-built AGC/AEC mechanisms and, in these circumstances, it can be necessary to deactivate AGC and AEC on Skype. If you disable AGC/AEC on Skype, ensure that the client defaults to enabled if the audio device is removed.

To query whether AGC and AEC are on:

Syntax

```
-> GET AGC
-> GET AEC
```

Response

```
<- AGC { ON | OFF }
<- AEC { ON | OFF }
```

To set AGC and AEC on and off:

Syntax

```
-> SET AGC ON | OFF
-> SET AEC ON | OFF
```

Response

```
<- AGC ON | OFF
<- AEC ON | OFF
```

Error codes

```
ERROR 569 - GET AEC: target not allowed
ERROR 570 - SET AEC: invalid value
ERROR 571 - GET AGC: target not allowed
ERROR 572 - SET AGC: invalid value
```

Version
Protocol 5

RESETIDLETIMER

This command resets the idle timer (the one that turns user's online status to "Away").

Note that there is currently no way of retrieving actual "Show my away when inactive for X minutes" setting from user profile. If you want to ensure the user status stays permanently online, it is sufficient to send `RESETIDLETIMER` every 59 seconds as it is impossible to set the auto-idle timer below 1 minute.

Syntax:
```
-> RESETIDLETIMER

<- RESETIDLETIMER
```

Version:
API version 3.2 (protocol 7)

GET AUTOAWAY

Returns the current state of automatic online status switcher.

Syntax:
```
-> GET AUTOAWAY

<- AUTOAWAY ON
```

SET AUTOAWAY

Sets the state of automatic online status switcher.

Syntax:
```
-> SET AUTOAWAY ON|OFF

<- AUTOAWAY ON|OFF
```

Example:

```
-> SET AUTOAWAY ON
<- AUTOAWAY ON
-> SET AUTOAWAY OFF
<- AUTOAWAY OFF
-> SET AUTOAWAY BANANA
<- ERROR 53 SET AUTOAWAY invalid value
```

Notifications

Notifications are sent by Skype if an object changes or if the value of a property is requested with a GET command. Also, if a property value is changed by a SET command, the change is confirmed with a notification. Notifications occur in the same manner, whether the related change is initiatied by the Skype UI or by an API client. There are two main types of notification:

- Object notifications occur when an object is created (for example due to an incoming call or chat), if an object changes, or if a property is queried.
- Status notifications are broadcast by Skype after an initial connection is made or if a parameter changes. These notifications can be queried at any time with the GET command.

Object notifications

This section contains the Skype object notifications.

Call notifications

Call notifications are sent on incoming calls or when an active calls changes. Clients can monitor call events to detect incoming calls and act on them (for example, to answer automatically).

Syntax
```
CALL <id> property <value>
```

Parameters
Refer to the CALL object for available properties and property values.

User notifications

User notifications are the most frequent notifications and include last-seen timestamps and user property information.

Syntax
```
USER <id> property <value>
```

Parameters
Refer to the USER object for available properties and property values.

Note
User notifications are reported also for users who are not in the contactlist which the client can ignore.

Chat notifications

Chat notification is sent when a chat is created, chat properties or members change, or a new message is posted into chat. A new message also triggers a chatmessage notification.

Syntax:
```
CHAT <id> property <value>
```

Parameters
Refer to the CHAT object for available properties and property values.

In version 3.6 additional notification messages were added on chat window open and close events.

Syntax:
```
CHAT <id> CLOSED|OPEN
```

Example:

```
<- CHAT #anappo2/$anappo;87ba791d4025455c CLOSED
<- CHAT #anappo2/$anappo;87ba791d4025455c OPENED
```

Chatmessage notifications

Chatmessage notification is sent when a new message arrives. The client can monitor these messages to display received messages.

Syntax
```
CHATMESSAGE <id> property <value>
MESSAGE <id> property <value>
```

Parameters
Refer to the CHATMESSAGE object for available properties and property values.

Notes
The MESSAGE command is deprecated in Protocol 3

Voicemail notifications

Voicemail notification is sent when a new voicemail is received or recorded.

Syntax
```
VOICEMAIL <id> property <value>
```

Parameters
Refer to the VOICEMAIL object for available properties and property values.

Application notifications

Application notifications are sent when a new application requests to connect, or when data is sent or received.

Syntax
```
APPLICATION <appname> property <value>
```

Parameters
Refer to the APPLICATION object for available properties and property values.

Status notifications

This section contains the Skype status notifications.

Callhistory change notification

This notification occurs when call history changes and needs to be reloaded. This change occurs when the call history or a selection of it has been deleted.

Syntax
```
CALLHISTORYCHANGED
```

Instant message history change

This notification occurs when instant message history changes and needs to be reloaded. It occurs only when all IM history is deleted.

Syntax
```
IMHISTORYCHANGED
```

Contactlist change notification

This notification occurs if a user is added to or deleted from contacts or has authorized the current user as a contact.

Syntax
```
USER <username> BUDDYSTATUS <status>
```

Parameters
Refer to the USER object for available status values.

Example

```
// User has been added to contacts, pending authorisation.
<- USER pamela BUDDYSTATUS 2
// User has authorized current user
<- USER pamela BUDDYSTATUS 3
// User has been deleted from contacts.
<- USER pamela BUDDYSTATUS 1
```

Contact group change notification

This notification is sent when GROUP USERS changes – when a user comes online or goes offline.

Syntax:
```
<- GROUP <group_id> NROFUSERS <n>
```

Example:

```
<- GROUP 56 NROFUSERS 19
<- USER test ONLINESTATUS OFFLINE
```

Version
Protocol 7 (API version 3.0)

User status notification

Syntax
`USERSTATUS status`

Parameters
`status` – value for user status. Possible values:

- `UNKNOWN` – no status information for current user.
- `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.

Connection status

Syntax
`CONNSTATUS status`

Parameters
`status` – value for connection status. Possible values:

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

Current user handle

Syntax
`CURRENTUSERHANDLE <username>`

Example
`CURRENTUSERHANDLE banana`

Contact list focus nofication

This notification occurs when contactlist focus changes:

Syntax

- `CONTACTS FOCUSED username` – when contact gains focus
- `CONTACTS FOCUSED` – when loses focus

Error codes

Skype sends an error response when it encounters an issue such as incorrect commands or internal inconsistencies. The error code is a number that uniquely identifies the error condition and the DESC is an optional brief description of the issue.

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	&nspb;
5	Search CALLS: invalid target	An unpermitted character (e.g. "!", "#", "$" etc.) was used in the target username.
6	SEARCH MISSEDCALLS: target not allowed	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.

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	Invalid protocol number (e.g. "PROTOCOL -12,9")
28	Unknown userstatus	Unknown or misspelled value for user status (e.g. "SET USERSTATUS RICH")
29	SEARCH what: target not allowed	Target is not permitted; 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	Property missing or misspelled
33	invalid parameter	Unknown or misspelled value for mute (e.g. "SET MUTE O")
34	invalid user handle	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".
50	cannot set device	An error occurred when changing audio device
51	invalid parameter	Parameter to READY command is not YES or NO
52	invalid parameter	Parameter to HOOK command is not ON or OFF. NB! HOOK command is no longer supported or relevant.
53	invalid value	Parameter to SET AUTOAWAY is not ON or OFF
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

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.
74	Invalid key	Key name in BTN_PRESSED or BTN_RELEASED command is invalid
91	call error	Cannot call an emergency number
92	call error	The called number is not a valid PSTN number
93	call error	Invalid Skype Name
94	call 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
108	User not contact	TRANSFER can only be initiated to contacts
109	directory doesn't exist	Directory given as a parameter to TRANSFER command does not exist
110	No voicemail capability	User given as a parameter to VOICEMAIL command doesn't have voicemail capability
111	File not found	File given as argument to SET AVATAR or SET RINGTONE command doesn't exist
112	Too many targets	Number of target users for OPEN FILETRANSFER command exceeds simultaneous filetransfer limit

114	Invalid avatar	GET or SET AVATAR avatar index invalid
115	Invalid ringtone	GET or SET RINGTONE ringtone index invalid
500	CHAT: Invalid chat name given	
501	CHAT: No chat found for given chat	
502	CHAT: No action name given	
503	CHAT: Invalid or unknown action	
504	CHAT: action failed	
505	CHAT: LEAVE does not take arguments	
506	CHAT: ADDMEMBERS: invalid/missing user handle(s) as arguments	
507	CHAT: CREATE: invalid/missing user handle(s) as argument	
508	CHAT: CREATE: opening a dialog to the given user failed	
509	No chat name given	
510	Invalid/uknown chat name given	
511	Sending a message to chat failes	
512	Invalid voicemail id	
513	Invalid voicemail object	
514	No voicemail property given	
515	Assigning speeddial property failed	
516	Invalid value given to ISAUTHORIZED/ISBLOCKED	
517	Changing ISAUTHORIZED/ISBLOCKED failed	
518	Invalid status given for BUDDYSTATUS	
519	Updating BUDDYSTATUS failed	
520	CLEAR needs a target	
521	Invalid/unknown CLEAR target	
522	CLEAR CHATHISTORY takes no arguments	
523	CLEAR VOICEMAILHISTORY takes no arguments	
524	CLEAR CALLHISTORY: missing target argument	
525	CLEAR CALLHISTORY: invalid handle argument	

526	ALTER: no object type given	
527	ALTER: unknown object type given	
528	VOICEMAIL: No proper voicemail ID given	
529	VOICEMAIL: Invalid voicemail ID given	
530	VOICEMAIL: No action given	
531	VOICEMAIL: Action failed	
532	VOICEMAIL: Unknown action	
534	SEARCH GREETING: invalid handle	
535	SEARCH GREETING: unable to get greeting	
536	CREATE: no object type given	
537	CREATE : Unknown object type given.	
538	DELETE : no object type given.	
539	DELETE : unknown object type given.	
540	CREATE APPLICATION : missing of invalid name.	
541	APPLICATION : Operation Failed.	
542	DELETE APPLICATION : missing or invalid application name.	
543	GET APPLICATION : missing or invalid application name.	
544	GET APPLICATION : missing or invalid property name.	
545	ALTER APPLICATION : missing or invalid action.	
546	ALTER APPLICATION : Missing or invalid action	
547	ALTER APPLICATION CONNECT: Invalid user handle	
548	ALTER APPLICATION DISCONNECT: Invalid stream identifier	
549	ALTER APPLICATION WRITE : Missing or invalid stream identifier	
550	ALTER APPLICATION READ : Missing or invalid stream identifier	
551	ALTER APPLICATION DATAGRAM : Missing or invalid stream identifier	
552	SET PROFILE : invalid property profile given	

	voicemail privledge, can't forward to voicemail.	
555	CALL: No proper call ID given	
556	CALL: Invalid call ID given"	
557	CALL: No action given	
558	CALL: Missing or invalid arguments	
559	CALL: Action failed	
560	CALL: Unknown action	
561	SEARCH GROUPS: invalid target"	
562	SEARCH GROUPS: Invalid group id	
563	SEARCH GROUPS: Invalid group object	
564	SEARCH GROUPS: Invalid group property given	
569	GET AEC: target not allowed"	
570	SET AEC: invalid value"	
571	GET AGC: target not allowed"	
572	SET AGC: invalid value"	
9901	Internal error	

Skype URI handler

Although not part of the Skype Desktop API, Skype 1.4 and later include a set of useful commands which can be initiated using the skype URI handler.

General syntax

```
SKYPE_URI    = "skype:" [targets] ["?" query ] ["#" fragment ]
targets      = 1* (target / ";" )
target       = identity / PSTN
identity     = skypename / alias
skypename    = 1*(ALPHA / DIGIT / "." / "," )
skypenames   =  1*( skypename / ";")
alias        =  ... ; see ["TechGroup/DataFormats"]
; unicode chars are in UTF-8 and % encoded; see RFC3987 uchar mapping
PSTN         = "+" (DIGIT / ALPHA ) *(DIGIT / ALPHA / "-" )   ; supports +800-FLOWERS
query        = action [ *( "?" term "=" conditon ) ]
term         = 1*ALPHA
condition    = 1*unserved   ; to be clarified
fragment     = 1*unserved   ; to be clarified
```

Skype for Windows 1.4 version handles the following

```
skype:                                      ; focus / open skype UI
skype:[targets]                             ; take default double-click action on contact
skype:[targets]?call                        ; call to target(s): can be skypename, alias or
PSTN
skype:[skypenames]?chat                     ; start chat/multichat with skypename(s)
skype:[skypename]?voicemail                 ; leave voicemail to skypename
skype:[skypename]?add                       ; add skypename to contactlist; show
authorization dialog
skype:[skypename]?add&displayname=customname ; add contact dialog with pre-set displayname
```

```
skype:[skypename]?userinfo                    ; show info (profile) for [username]
skype:[skypename]?chat&topic=[topic]          ; opens chat with pre-set topic;
skype:?chat&amp;id=[id][#time]                ; open existing multichat with [id];
; time: YYYY-MM-DDThh:mm:ssTZ / YYYY-MM-DDZhh:mm:ss
```

Examples

- [skype:echo123 skype:echo123]
- [skype:echo123?call skype:echo123?call]
- [skype:echo123?chat skype:echo123?chat]
- [skype:echo123?chat&topic=Test skype:echo123 chat with pre-set topic]
- [skype:echo123?add&displayname=Skype%20Test%Call skype:echo123 add contact as Skype Test Call]

Notice that there is no "//" in skype: URI – `skype://echo123 does` **not** work.

Release Notes

Skype 4.0 GOLD Release Notes

Date: 2009-01-22

While the Windows 4.0 client release does not bring any new features to the Desktop API, it does present a major UI overhaul. As a result, some parts of the Desktop API that had dependencies in the UI are no longer functional – the corresponding UI parts having been removed or not yet implemented. We have tried to keep the list of nonfunctional Desktop API commands down to minimum. Also note that the commands do not fail with error messages – they just have no effect in the UI.

Here is the list:

Skype Alert Events

The entire custom events system is currently unavailable.

- `CREATE EVENT` – unavailable.
- `DELETE EVENT` – unavailable.

Skype Custom Menus

In 3.x the create menu command had four different contexts, where the custom menu Items could be created in: CALL, MYSELF, TOOLS, CONTACT. Currently only TOOLS context remains functional.
*

- `CREATE MENU_ITEM <ID> CONTEXT CALL CAPTION <text>` – unavailable.
- `CREATE MENU_ITEM <ID> CONTEXT MYSELF CAPTION <text>` – unavailable.
- `CREATE MENU_ITEM <ID> CONTEXT CONTACT CAPTION <text>` – unavailable.

Send Contacts

- `OPEN SENDCONTACTS` – unavailable.

Skype 3.6. Release Notes

Date: 2007-10-03

New notification messages:

- `CHAT <id> CLOSED|OPEN` – notifications added on chat window open and close events. See Chat

New commands:

- GET WINDOWSTATE – returns current state of the Skype main window.
- SET WINDOWSTATE – sets state of the Skype main window.

Varia:

As of ths version various commands no longer accept "¤", "€" or "£" symbols in their parameters. Instead, `ERROR 8 Invalid user handle` error message is generated in response. Following commands are affected:

- `CALLVOICEMAIL <handle>`
- `GET USER <handle> HANDLE`
- `SEARCH CALLS <handle>`
- `SEARCH CHATMESSAGES <handle>`
- `SEARCH MESSAGES <handle>`

Skype 3.5.0.202 Release Notes

Date: 2007-08-07

New VOICEMAIL object properties:

- INPUT
- OUTPUT
- CAPTURE_MIC

VOICEMAIL audio stream access commands:

- ALTER VOICEMAIL SET_INPUT
- ALTER VOICEMAIL SET_OUTPUT
- ALTER VOICEMAIL SET_CAPTURE_MIC

Skype 3.5 Release Notes

Date: 2007-07-02

New protocol version: 8

- New CALL STATUS enumerator – `WAITING_REDIAL_COMMAND`.
- New CALL STATUS enumerator – `REDIAL_PENDING`.
- New SMS FAILUREREASON enumerator – `NO_SENDERID_CAPABILITY`.
- Sending chat messages and `CHAT CREATE` commands may now fail with a new error code: 615, "CHAT: chat with given contact is disabled".
- When the UI language is set via custom language file, `GET UI_LANGUAGE` will return "xx" (used to return "en" in versions prior to 3.5).

Skype 3.2 Release Notes

Date: 2007-04-30

New commands:

- OPEN LIVETAB

New USER object property:

- IS_VOICEMAIL_CAPABLE

Skype 3.1 Release Notes

Date: 2007-04-05

New commands:

- GET AVATAR
- SET AVATAR
- GET USER AVATAR
- GET PREDICTIVE_DIALER_COUNTRY
- GET CONTACTS FOCUSED
- GET RINGTONE STATUS

New CALL property – TARGET_IDENTITY.

New CHAT property – TOPICXML.

Error reporting changed for SET VIDEO_IN command.

Skype 3.0 Release Notes

New protocol version: 7

Support for custom menus

Refer to Custom Menu Items section for more information.

Support for custom events

Refer to Skype Alert Events for more information.

Call transfer API

New commands and object properties to support call transfers:

- GET CALL CAN_TRANSFER
- ALTER CALL TRANSFER

New CALL statuses:

- TRANSFERRING
- TRANSFERRED

New call transfer related CALL properties:

- TRANSFER_STATUS
- TRANSFER_ACTIVE
- TRANSFERRED_BY
- TRANSFERRED_TO

File transfer object

Refer to FILETRANSFER object section for more information.

Notification changes

- USER LASTONLINETIMESTAMP – events about its change are no longer sent to clients, the property can still be queried.
- GROUP NROFUSERS_ONLINE – events about its change are no longer sent to clients, the property can still be queried.
- GROUP USERS – events about its change are no longer sent to clients – instead GROUP NROFUSERS event is generated; if you get an event on NROFUSERS you can assume the GROUP USERS has changed.
- GROUP NROFUSERS – is now only sent when GROUP USERS property changes.

Richtext mood messages

New property RICH_MOOD_TEXT was added to PROFILE and USER objects.

New moodmessage related commands are:

- SET PROFILE RICH_MOOD_TEXT
- GET USER RICH_MOOD_TEXT

Wallpapers

New GET WALLPAPER and SET WALLPAPER commands. Refer to [#COMMAND_UI_WALLPAPERS GET/SET WALLPAPERS] section.

Public chats

New CHATMEMBER object.

New CHAT object properties:

- MEMBEROBJECTS
- PASSWORDHINT
- GUIDELINES
- OPTIONS
- DESCRIPTION
- DIALOG_PARTNER
- ACTIVITY_TIMESTAMP
- TYPE
- MYSTATUS
- MYROLE
- BLOB
- APPLICANTS

New CHATMESSAGE properties:

- EDITED_BY
- EDITED_TIMESTAMP
- IS_EDITABLE
- OPTIONS
- ROLE

Modified CHATMESSAGE property TYPE enumerations for PROTOCOL 7:

- POSTEDCONTACTS
- GAP_IN_CHAT
- SETROLE
- KICKED
- SETOPTIONS
- KICKBANNED
- JOINEDASAPPLICANT
- SETPICTURE
- SETGUIDELINES

The BODY property of a CHATMESSAGE object is now read-write. Refer to SET CHATMESSAGE BODY command for more information.

New CHATMEMBER related commands are:

- ALTER CHATMEMBER SETROLETO
- ALTER CHATMEMBER CANSETROLETO

New CHAT commands:

- ALTER CHAT JOIN
- ALTER CHAT KICK
- ALTER CHAT KICKBAN
- ALTER CHAT DISBAND
- ALTER CHAT ENTERPASSWORD
- ALTER CHAT CLEARRECENTMESSAGES
- ALTER CHAT ACCEPTADD
- ALTER CHAT SETOPTIONS
- ALTER CHAT SETGUIDELINES
- ALTER CHAT SETALERTSTRING
- ALTER CHAT SETPASSWORD
- CHAT FINDUSINGBLOB
- CHAT CREATEUSINGBLOB
- ALTER CHAT SETDESCRIPTION

CHAT CREATE no longer requires usernames, if you provide no usernames a general multichat is created.

Change in text value parsing: all texts which include whitespace must be quoted.

Skype 2.6 Release notes

Voice API

New CALL object properties:

- INPUT
- OUTPUT

New Voice API related commands:

- GET CALL <INPUT|OUTPUT>
- GET CALL CAPTURE_MIC
- ALTER CALL SET_<INPUT|OUTPUT>
- ALTER CALL SET_CAPTURE_MIC

Refer to Voice Streams section for more information.

SMS API

New object: SMS

New SMS related commands:

- GET SMS CHUNK
- GET SMS CHUNKING
- SET SMS SEEN
- SET SMS BODY
- SET SMS REPLY_TO_NUMBER
- SET SMS TARGET_NUMBERS
- CREATE SMSS
- ALTER SMS SEND
- DELETE SMSS
- SEARCH SMSS
- SEARCH MISSEDSMSS
- GET PROFILE SMS_VALIDATED_NUMBERS

Shared contact groups

New GROUP object types, (protocol 6):

- SHARED_GROUP
- PROPOSED_SHARED_GROUP

New commands related to shared groups:

- ALTER GROUP SHARE
- ALTER GROUP ACCEPT
- ALTER GROUP DECLINE

Refer to ALTER GROUP SHARE command for more information.

Call cost information

New CALL object properties

- RATE
- RATE_CURRENCY
- RATE_PRECISION

Refer to Call cost information section for more information.

Chat Bookmarks

New CHAT object property: BOOKMARKED

New commands related to shared groups:

- ALTER CHAT BOOKMARKED
- ALTER CHAT UNBOOKMARK

Refer to ALTER CHAT BOOKMARKED section for more information.

Various new object properties:

- New USER object property: NROF_AUTHED_BUDDIES
- New CALL object property: FORWARDED_BY
- New CALL object property: VAA_INPUT_STATUS

Various new commands:

- SET SILENT_MODE
- SEARCH MISSEDVOICEMAILS
- GET PROFILE IPCOUNTRY – refer to PROFILE object for more information.

GET_CONFERENCE_PARTICIPANT_COUNT now reports the number of conference call participants more correctly.

VOICEMAIL command enters the deprecation process and is replaced by new command: CALLVOICEMAIL

PONG reply to PING is now asynchronous.

Skype 1.4 Release Notes

Date: 2005-09-16
Changes and fixes:

- Support for application to application messaging
- Set profile properties
- Support for call forwarding
- Extended support to open client windows
- New user object properties (mood text, alias)
- Extended support for ringtones
- Support for Skype URI handler commands
- Support for contact focused notifications

Skype 1.3.0.42 release notes

Date: 2005-06-11
Changes and fixes:

- added: Protocol 5
- Support for voicemails: `VOICEMAIL, OPEN VOICEMAIL, ALTER VOICEMAIL, SEARCH VOICEMAILS`
- Support for chat handling: `CHAT CREATE, OPEN CHAT, ALTER CHAT, SEARCH *CHATS`
- Support for authorizations: `SEARCH USERSWAITINGMYAUTHORIZATION, SET USER, ISAUTHORIZED, ISBLOCKED, BUDDYSTATUS`
- Support for deleting history: `CLEAR CHATHISTORY, VOICEMAILHISTORY, CALLHISTORY`
- Set ringing device: `SET/GET RINGER`
- Extended `DTMF` support: `SET CALL DTMF`
- Initiate filetransfer: `OPEN FILETRANSFER`
- Assign speeddial: `USER SPEEDDIAL`
- Change ringtones: `GET/SET RINGTONE`
- Change avatar: `SET AVATAR`
- Minimize Skype window: `MINIMIZE`
- bugfix: conference call bugs resolved

Skype 1.2.0.11 release notes

Date: 2005-03-04
Changes and fixes:

- added: Protocol 4
- Support for conferencing: start a conference, add people to conference and being able to get list of conference call participants and notifications about these
- Possible 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: from protocol 4 language and country values are prefixed by ISO codes, for example `'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 userhandle to `OPEN ADDAFRIEND`
- Support for command-id (`#1 SET xxx`)
- `CALL FAILUREREASON 1` – documentation error, 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`
- change: Up/down via phone api autoexpand contactlist groups
- bugfix: `"AUDIO IN"` and `"AUDIO OUT"` commands do nor read double byte driver names correctly
- bugfix: `BTN_PRESSED E` fails with error 71 invalid key
- bugfix: Muting microphone in UI not reflected in API
- bugfix: Conference to more than 4 participants causes "Range check" errors
- bugfix: `IMHISTORYCHANGED` doesn't work
- bugfix: `SET MUTE ON` returns always `MUTE OFF`
- bugfix: Cannot call SkypeOut contacts using speeddial
- bugfix: No response to empty `CALL` (should return ERROR 34 invalid user handles
- bugfix: Skype access control does not deny access to a device
- bugfix: No notification if the user changes audio device

Skype 1.1.0.61 release notes

Date: 2005-01-12
Changes and fixes:

- added: Protocol 3
- change: API – now allows one ongoing search per client only. Attempting to issue new search before receiving results of a previous one results in error 72.

- 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 access control list only one program's permission was remembered
- bugfix: Multichat message IDs were not returned
- bugfix: Problems with connecting for older applications
- bugfix: Fixed API exceptions if Skype is used on two Windows accounts simultaneously
- bugfix: On Windows98/ME some dll files were shown to use Skype instead of the respective application
- bugfix: Sometimes API didn't return 'BUDDYSTATUS 1' messages

Skype 1.0.0.94 release notes

Date: 2004-10-21
Changes and fixes:

Release of Skype Desktop (Public) API.