TrueConf

TrueConf Chatbot Connector

Developer Guide for Chat Bots

Table of Contents

1. General Overview

TrueConf Server API for Chat Bots enables the integration of bots into the existing infrastructure and enhances the functionality of client applications.

i This API is included with TrueConf Server starting from version 5.5.0.

We covered the features of in detail in our webinar:

- https://youtu.be/GDvs6oN7_b8

1.1. Features

Currently, the following features are available in TrueConf Server API for Chat Bots:

- Creating personal and group chats and channels.
- Sending messages with HTML and Markdown formatting.
- File transfer.
- Polls submission.
- Forwarding messages and replying to messages.
- Editing and deleting messages.
- Retrieving information about chats (channels), messages, and files.
- Retrieving the list of chats and channels.
- Retrieving chat (channel) history.
- Checking participant presence in the chat.
- Retrieving the list of chat (channel) participants.
- Adding a participant to a group chat and channel.
- Removing a participant from a group chat and channel.
- Retrieving the display name of the user.

The following event notifications are available:

- A new personal chat / group chat / channel has been created.
- Participant added/removed from chat (channel)
- A new message has been sent to the chat.
- File sent to chat
- A poll has been sent to the chat
- The message has been edited.
- The message has been deleted

2. Connection and Authorization

TrueConf Server includes the TrueConf Bridge service. This service contains an HTTP and WebSocket server used to implement API functionality for chat operations. The TrueConf Bridge service uses TCP port 4309.

To send API requests (commands), you must first connect to the WebSocket server and authorize the connection using an access token and the auth request.

To connect to the server, you need to establish a WebSocket connection at the following address: `/weboscket/chat_bot/` :

From localhost directly to the TrueConf Bridge service (port 4309) using WebSocket (ws):

```
ws://localhost:4309/websocket/chat_bot/
```

From outside to the TrueConf Web service (port 443 or other) using WebSocket Secure (wss):

```
wss://video.example.net/websocket/chat_bot/
```

The server supports the **json.v1** subprotocol, which is transmitted in the **Sec-WebSocket-Protocol** header. If the header is not specified, json.v1 will be used by default.

2.1. Supported Accounts

Only user accounts from the current TrueConf server are supported. Authorization through accounts from federated servers or guest accounts is not supported. Accounts are supported in both Registry and LDAP modes.

2.2. Identification token

Identification token is required for authorizing a WebSocket connection. The token is obtained via an HTTP request to the `/bridge/api/client/v1/oauth/token` endpoint following the OAuth 2.0 standard. Note that the TrueConf Bridge service does not accept HTTPS requests. For information on how to obtain a token using an encrypted connection (HTTPS), see below.

2.2.1. Token request

To obtain an identification token, execute a POST request:

```
POST /bridge/api/client/v1/oauth/token HTTP/1.1
Host: localhost:4309
Content-Type: application/json
Content-Length: 114
```

```
{
    "client_id": "chat_bot",
    "grant_type": "password",
    "username": "user",
    "password": "qwerty"
}
```

Parameter	Description
client_id	Constant string `"chat_bot"`
grant_type	Constant string `"password"`
username	User's login (see here)
password	User's password

2.2.2. Successful authorization

If the authorization is successful, the HTTP server will respond with the code `201 Created`:

```
HTTP/1.1 201 Created
Content-Type:application/json
Access-Control-Allow-Origin:*
Date:Mon, 09 Sep 2024 22:42:50 GMT

{
    "access_token":
"eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ",
    "token_type": "JWE",
    "expires_in": 31536000
}
```

Parameter	Description
access_token	An identification token used for authorizing a WebSocket connection.
token_type	A constant string `"JWE"`.
expires_in	The token's lifespan is **one year** from the time of its creation. After this period, re-authorization will be necessary.

Read more about the returned structure in the response body (payload) here.

2.2.3. Authorization Error

In case of an authorization error, the HTTP server will respond with a `400 Bad Request` code:

```
HTTP/1.1 400 Bad Request
Content-Type:application/json
Access-Control-Allow-Origin:*
Date: Mon, 09 Sep 2024 22:42:50 GMT

{
"error": "invalid_grant",
"error_description": "Invalid username or password"
}
```

Parameter	Description
error	Value from the OAUTH_ERROR enumeration.
error_description	Human-readable text of the authorization error.

Read more about the return structure in the response body (payload) here.

2.2.4. Using HTTPS

HTTP protocol is used for:

- obtaining an identification token;
- uploading files to a dedicated storage on the server.

The TrueConf Bridge service does not support the HTTPS protocol. If you need to use HTTPS, requests must be directed to the TrueConf web server implemented by the TrueConf Web Manager service. The TrueConf Web Manager service proxies all HTTPS requests received at the `/bridge/api/` endpoint to the TrueConf Bridge service.

Thus, when it is necessary to use the HTTPS protocol, requests can be sent to port 443 or another port, as specified in the control panel of TrueConf Server, after ensuring that the TrueConf Web Manager service is running:

```
https://video.example.net/bridge/api/client/v1/oauth/token
```

2.3. WebSocket Connection Authorization

After obtaining the identification token, you need to authorize your WebSocket connection.

Request:


```json
{
  "type": 1,
  "id": 1,
  "method": "auth",
  "payload": {
    "token": "someToken",
    "tokenType" : "JWE"
  }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incremented value assigned by the sender, mandatory in each request for subsequent binding to the response. More details can be found here
method	*string*	Yes	Type of command being sent, default is `auth`
token	*string*	Yes	Authorization token, obtained via HTTP request
tokenType	*string*	Yes	Default is `JWE`

Answer:

```json
{
  "type": 2,
  "id": 1,
  "payload": {
    "userId": "someTrueConfUserId@someTrueConfServeName/someHash"
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to match the request and response
userId	*string*	Yes	TrueConf ID of the authorized bot (server user)

> *i* In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

2.4. Sending messages without authorization

If you start sending messages over a WebSocket without executing the `auth` command, any message will receive an error with code `200`.

3. Basic WebSocket Server Message Format

WebSocket is a bidirectional transport: both parties can send any set of data in the **JSON** format at any time. The protocol implements a "**request-response**" mechanism, where each incoming message must be acknowledged by the recipient.

Message Types

The protocol defines three types of messages corresponding to the MESSAGE_TYPE enumeration:

- RESERVED = 0 – reserved (not in use);
- REQUEST = 1 – request;
- RESPONSE = 2 – response.

* Currently, only REQUEST and RESPONSE are used.

Any message, whether from the client or the server, contains at least two mandatory fields:

```json
{
  "type": 1,
  "id": 1,
}
```

Field	Type	Req.	Description
type	uint32	Yes	Message type: `1` for request (REQUEST), `2` for response (RESPONSE).
id	uint32	Yes	A unique message identifier, represented as an incrementing number. Used to link request-response pairs. For more details on how to use it, see below.

3.1. Server response

Within the operation of WebSocket, the server can initiate the sending of system messages without a prior request from the client. These messages notify about changes, system events, or actions of other users.

To maintain a stable connection, the client must acknowledge the receipt of each such message. The acknowledgment is sent as a separate response indicating the original event identifier.

```json
{
  "type": 2,
```

```
  "id": 123456
}
```

If the server does not receive a response within 300 seconds, the connection is considered inactive and will be automatically closed, regardless of other signs of activity (e.g., ping/pong).

3.2. Working with the id parameter

Since WebSocket does not have a request-response concept and the order of incoming messages can be arbitrary, the `id` parameter is used to link outgoing commands with their responses. This parameter must be included in every sent request, and the same value will appear in the incoming message that serves as the response to the request. It can also be interpreted as a counter of sent requests for each party. A response with the same `id` must be sent for each incoming request.

For example, the client sends a `ping` message to the server:

```json
{
"type": 1,
"id": 1,
"method": "ping",
"payload": {
  "data": "Hello, World!"
  }
}
```

The server responds to the client (note the `id` parameter):

```json
{
  "type": 2,
  "id": 1,
  "payload": {
    "some_field": true
  }
}
```

Another example. The client sends a message to the server:

```json
{
  "type": 1,
  "id": 23768,
```

```
  "method": "getCurrentTime"
}
```

The server responds:

```json
{
  "type": 2,
  "id": 23768,
  "payload": {
    "data": 1724936321584
  }
}
```

3.3. Timeout and long operations

If a response to the request is not received within 300 seconds (5 minutes), the server will assume that the connection has been lost or the client has become unresponsive, and will close the WebSocket session. For certain tasks, generating a response may take more than 300 seconds. In such cases, multiple request-response cycles will be generated.

As an example, let's consider the call handling logic. A person may not answer (reject) a call for more than 300 seconds. In such a case, we receive the call status as a separate message from the server after any length of time:

Client request:

```json
{
  "type": 1,
  "id": 4444,
  "method": "callToUser",
  "payload": {
    "callId": "john"
  }
}
```

The server immediately responds with a call creation confirmation:

```json
{
  "type": 2,
  "id": 1,
  "payload": {
    "userId": "john@as1.trueconfServer.loc#vcs"
```

```
  }
}
```

After some time, when the user answers the call, the server sends a notification to the client:

```json
{
  "type": 1,
  "id": 4445,
  "method": "conferenceCreated",
  "payload": {
    "userId": "john@as1.trueconfServer.loc#vcs",
    "isP2P": true,
    "conferenceId": "477dh6731ac54e322@as1.trueconfServer.loc#vcs"
  }
}
```

The client confirms that it has received the conference start event:

```json
{
  "type": 2,
  "id": 4445
}
```

3.4. Duplicate IDs in requests

If we send two requests with the same `id`, the second request will be rejected, and an error will be returned in response. Similarly, if the `id` value in a new request is less than an already used value, that request will also be rejected with an error.

Request:

```json
{
  "type": 1,
  "id": 12,
  "method": "ping"
}
```

Answer:

```json
{
  "type": 2,
```

```
  "id": 12
}
```

Resending request with `id` = 12:

```json
{
  "type": 1,
  "id": 12,
  "method": "someTest"
}
```

Error response with `errorCode` = 2:

```json
{
  "type": 2,
  "id": 12,
  "payload": {
    "errorCode": 2
  }
}
```

3.5. Asynchronous execution of requests

Since WebSocket libraries provide asynchronous APIs, the client or server can send responses to requests in any order and multiple at a time, without waiting for a response to a previous request. The key point is that the request must be sent within 5 minutes:

Request 1:

```json
{
  "type": 1,
  "id": 278,
  "method": "someLongTask"
}
```

Request 2:

```json
{
  "type": 1,
  "id": 279,
  "method": "createChat",
```

```
  "payload": {
    "chatTitle": "TEST"
  }
}
```

Request 3:

```json
{
  "type": 1,
  "id": 280,
  "method": "createChat",
  "payload": {
    "chatTitle": "TEST 2"
  }
}
```

Request 4:

```json
{
  "type": 1,
  "id": 281,
  "method": "someFastTask"
}
```

Response 1 to request 4:

```json
{
  "type": 2,
  "id": 281
}
```

Response for request 2:

```json
{
  "type": 2,
  "id": 279,
  "payload": {
    "chatId": "1390r4f03f"
```

```
  }
}
```

Response 3 to request 3:

```json
{
  "type": 2,
  "id": 280,
  "payload": {
    "chatId": "1f4f9f9f39j9f3"
  }
}
```

Response 4 to request 1:

```json
{
  "type": 2,
  "id": 278
}
```

3.6. Standard error response

For most requests, in case of an error, a response will be sent in the following format:

```json
{
  "type": 2,
  "id": 1,
  "payload": {
    "errorCode": 200
  }
}
```

Category	Code	Name	Description
Network Errors	100	CONNECTION_ERROR	Connection error
	101	CONNECTION_TIMEOUT	Connection timeout
	102	TLS_ERROR	TLS/SSL error
	103	UNSUPPORTED_PROTOCOL	Unsupported protocol
	104	ROUTE_NOT_FOUND	Route not found

Category	Code	Name	Description
Authorization Errors	200	NOT_AUTHORIZED	Not authorized
	201	INVALID_CREDENTIALS	Invalid credentials
	202	USER_DISABLED	User disabled
	203	CREDENTIALS_EXPIRED	Credentials expired
Chat Errors	300	INTERNAL_ERROR	Internal server error
	301	TIMEOUT	Operation timeout
	302	ACCESS_DENIED	Access denied
	303	NOT_ENOUGH_RIGHTS	Not enough rights
	304	CHAT_NOT_FOUND	Chat not found
	305	USER_IS_NOT_CHAT_PARTICIPANT	User is not a chat participant
	306	MESSAGE_NOT_FOUND	Message not found
	307	UNKNOWN_MESSAGE	Unknown message
	308	FILE_NOT_FOUND	File not found
	309	USER_ALREADY_IN_CHAT	User already in chat

3.7. Message Structure

Each message received as an event (server request), retrieved by message ID, or found in the chat message history corresponds to an Envelope object:

```json
{
    "method": "sendMessage",
    "type": 1,
    "id": 3,
    "payload": {
        "chatId": "1c1230635432aa7be051e4fda53a3d5a07c8c151",
        "messageId": "dfb127d0-d174-4e11-8394-19482a98607d",
        "timestamp": 1741881175593,
        "author": {
            "id": "user@video.example.com",
            "type": 1
        },
        "isEdited": false,
        "box": {
```

```
            "id": 2,
            "position": "0"
        },
        "type": 200,
        "content": {
            "text": "What's up?",
            "parseMode": "text"
        }
    }
}
```

* You can find the description of the parameters in this section.

3.8. Message Storage "Box"

According to TrueConf's internal protocol, each message is placed in a separate storage - a Box, also known as an "envelope" (object EnvelopeBox):

```json
{
"id" : 1,
"position" : ""
}
```

In most cases, there will be only one message in the box, and to properly sort the messages, it is sufficient to consider the `id` parameter. The `id` field is an automatically incrementing number, starting from zero within each chat. In the vast majority of cases, the first message in a chat will have a Box with `id` = 0, the second with `id` = 1, and so on. In this case, the `position` field for each message will be an empty string.

However, in some cases, a single "box" may receive two or more messages. This is a very rare occurrence, possible, for example, when two or more users simultaneously send messages to the chat, and both of those users experience network connection issues.

If two or more messages end up in the same "box," the `position` field is filled with the ordinal number of that message in the "box." The distinctive feature is that the position of the message in the box is not a number but a string containing the characters `0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz{}`.

To sort messages by `position`, it is sufficient to compare two strings based on their ASCII codes. This type of comparison is available by default in most programming languages.

Comparison of three messages that ended up in the same "box":

```json
//Message 1:
{
"id" : 15,
"position" : "A"
}
//Message 2:
{
"id" : 15,
"position" : "B"
}
//Message 3:
{
"id" : 15,
"position" : "AAA"
}
```

In "standard" string comparison, the ASCII value of each character in the strings is compared sequentially. The string with the greater ASCII value is considered greater. If the ASCII values of all characters are equal, then the string with the greater number of characters is considered greater.

Compare the first and second message:

```
ASCII("A") = 41
ASCII("B") = 42
42 > 41
```

Thus, the second message has a sequence number greater than the first.

Compare the second and third messages:

```
ASCII("B") = 66
ASCII("AAA") = 65 65 65
66 < 65
```

As can be seen, the check does not proceed beyond the first character since the character `B` has a higher ordinal number than the character `A`. Therefore, the third message has a lower ordinal number than the second.

Compare the first and third messages:


```
ASCII("A") = 41
ASCII("AAA") = 41 41 41
41 = 41
```

The ordinal numbers of characters in both strings match, and in this case, the string with the greater number of characters is considered larger. Thus, the third message has a higher ordinal number than the first.

After sorting, the order of messages will be as follows:

```json
//Message 1:
{
  "id": 15,
  "position": "A"
}
//Message 3:
{
"id" : 15,
"position" : "AAA"
}
//Message 2:
{
"id" : 15,
"position" : "B"
}
```

4. Participant rights and roles

4.1. Roles description

TrueConf Server supports the following roles within the chat:

- `owner` - group chat owner;
- `admin` - appointed moderator of the group chat;
- `user` - regular participant of a group chat;
- `conf_owner` — a role assigned to the owner of the conference; used only in conference chats.
- `conf_moderator` — the role of the conference moderator (operator); used exclusively in conference chats.
- `favorites_owner` — the role of the **Favorites** chat owner; used only in this type of chat.
- `writer` - the role of a user who has permission to write in a chat of type **channel**; used only in this context. type of chat.

* You can obtain the list of chat participants along with their roles using the getChatParticipants request.

4.2. Role permissions in a regular chat

	owner	admin	user	conf_owner	conf_moderator	favorites_owner
Reading messages	✅	✅	✅	✅	✅	✅
Sending messages	✅	✅	✅	✅	✅	✅
Adding participants to chat	✅	✅	❌	❌	❌	❌
Removing participants from chat	✅	✅	❌	❌	❌	❌
Editing own messages	✅	✅	✅	✅	✅	✅
Deleting own messages for oneself	✅	✅	✅	✅	✅	✅
	✅	✅	✅	✅	✅	✅

	owner	admin	user	conf_owner	conf_moderator	favorites_owner
Deleting own messages for all						
Deleting others' messages for oneself	✅	✅	✅	✅	✅	❌
Deleting others' messages for all	✅	✅	❌	✅	✅	❌
Clearing chat history for oneself	✅	✅	✅	✅	✅	✅
Clearing chat history for all	✅	✅	❌	✅	✅	❌
Replying to messages	✅	✅	✅	✅	✅	✅
Forwarding messages to/from chat	✅	✅	✅	✅	✅	✅
Sending files	✅	✅	✅	✅	✅	✅
Sending geolocation	✅	✅	✅	✅	✅	✅
Sending contacts	✅	✅	✅	✅	✅	✅
Sending emoji	✅	✅	✅	✅	✅	✅
Sending stickers	✅	✅	❌	✅	✅	✅
Pinning messages	✅	✅	❌	✅	✅	✅
Changing chat name	✅	✅	❌	❌	❌	❌

	owner	admin	user	conf_owner	conf_moderator	favorites_owner
Changing chat icon	✅	✅	❌	❌	❌	❌
Changing chat participant roles	✅	✅	❌	✅	✅	❌
Granting rights to chat participants	✅	✅	❌	✅	✅	❌
Revoking chat participant rights	✅	✅	❌	✅	✅	❌
Blocking participants	✅	✅	❌	❌	❌	❌
Changing chat owner	✅	❌	❌	❌	❌	❌
Setting the first visible message	✅	✅	❌	❌	❌	✅
Sending polls	✅	✅	✅	❌	❌	❌

4.3. Role permissions in the channel

	owner	admin	user	conf_owner	conf_moderator	favorites_owner
Reading messages	✅	✅	✅	✅	✅	✅
Sending messages	✅	✅	❌	✅	✅	✅
Adding participants to chat	✅	✅	❌	❌	❌	❌
Removing participants from chat	✅	✅	❌	❌	❌	❌
	✅	✅	❌	✅	✅	✅

	owner	admin	user	conf_owner	conf_moderator	favorites_owner
Editing own messages						
Deleting own messages for oneself	✅	✅	❌	✅	✅	✅
Deleting own messages for everyone	✅	✅	❌	✅	✅	✅
Deleting others' messages for oneself	✅	✅	❌	✅	✅	❌
Deleting others' messages for everyone	✅	✅	❌	✅	✅	❌
Clearing chat history for oneself	✅	✅	❌	✅	✅	✅
Clearing chat history for everyone	✅	✅	❌	✅	✅	❌
Replying to messages	✅	✅	❌	✅	✅	✅
Forwarding messages in/out of chat	✅	✅	✅	✅	✅	✅
Sending files	✅	✅	❌	✅	✅	✅
Sending location	✅	✅	❌	✅	✅	✅
Sending contacts	✅	✅	❌	✅	✅	✅

	owner	admin	user	conf_owner	conf_moderator	favorites_owner
Sending emojis	✅	✅	❌	✅	✅	✅
Sending stickers	✅	✅	❌	✅	✅	✅
Pinning messages	✅	✅	❌	✅	✅	✅
Changing chat title	✅	✅	❌	❌	❌	❌
Changing chat icon	✅	✅	❌	❌	❌	❌
Changing chat participants' roles	✅	✅	❌	❌	❌	❌
Granting rights to chat participants	✅	✅	❌	✅	✅	❌
Revoking chat participant rights	✅	✅	❌	✅	✅	❌
Blocking participants	✅	✅	❌	❌	❌	❌
Changing chat owner	✅	❌	❌	❌	❌	❌
Setting the first visible message	✅	✅	❌	❌	❌	✅
Sending polls	✅	✅	❌	❌	❌	❌

5. Working with Chats

5.1. Creating a personal chat with a user

Creating a personal chat (peer-to-peer) with a server user. If the bot has never messaged this user before, a new chat will be created. If the bot has previously sent messages to this user, the existing chat will be returned.

Request:

```json
{
    "type": 1,
    "id": 1,
    "method": "createP2PChat",
    "payload": {
        "userId": "user@video.example.com"
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Type of message (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. Incremental value assigned by the sender, required in each request for subsequent linking with the response. Read more here
method	*string*	Yes	Command `createP2PChat`
userId	*string*	Yes	TrueConf ID of the user with whom we want to create a chat

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": {
        "chatId": "4a1f88f1070d2f43d385cde9ff61964bc6b74477"
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to associate the request and response
chatId	*string*	Yes	Identifier of the created chat

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

5.2. Creating a group chat

Request:

```json
{
    "type": 1,
    "id": 1,
    "method": "createGroupChat",
    "payload": {
        "name": "TrueConf Server Administrators' Chat"
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required in each request for subsequent association with the response. Read more here
method	*string*	Yes	Command `createGroupChat`
name	*string*	Yes	Name of the group chat

Answer:

```json
{
    "type": 2,
```

```
    "id": 1,
    "payload": {
        "chatId": "2b896a6ef210d6b2dcaebc2186c9a7974d616054"
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to link the request and response
chatId	*string*	Yes	The identifier of the created group chat

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

5.3. Creating a Channel

Request:

json

```
{
    "type": 1,
    "id": 1,
    "method": "createChannel",
    "payload": {
        "title": "Company news"
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required in each request for subsequent matching with the response. Read more here
method	*string*	Yes	The command `createChannel`

Parameter	Type	Required	Description
title	*string*	Yes	Channel name

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": {
        "chatId": "2b896a6ef210d6b2dcaebc2186c9a7974d616054"
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier that matches the number sent in the original request, used for linking the request and response
chatId	*string*	Yes	Identifier of the created channel

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

5.4. Retrieving chat information by ID

Request:

```json
{
    "type": 1,
    "id": 1,
    "method": "getChatByID",
    "payload": {
        "chatId": "2b896a6ef210d6b2dcaebc2186c9a7974d616054"
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incremented value assigned by the sender, required in each request for subsequent linking to the response. Read more here
method	*string*	Yes	Command `getChatByID`
chatId	*string*	Yes	Chat identifier

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": {
        "chatId": "2b896a6ef210d6b2dcaebc2186c9a7974d616054",
        "title": "TrueConf Server Administrators' Chat",
        "chatType": 2,
        "unreadMessages": 0,
        "lastMessage": {
            "messageId": "267d61f2-2ba1-4e88-83ab-bb84415f31b6",
            "timestamp": 1746028006345,
            "author": {
                "id": "brown@video.example.com",
                "type": 1
            },
            "type": 200,
            "content": {
                "text": "What's up?",
                "parseMode": "html"
            }
        }
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE

Parameter	Type	Required	Description
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to link the request and response
chatId	*string*	Yes	Chat identifier
title	*string*	Yes	Chat title
chatType	*ChatTypeEnum*	Yes	Type of chat
unreadMessages	*uint32*	Yes	Number of unread messages in the chat
lastMessage	*Envelope*	Yes	Last message in the chat

* The `lastMessage` field follows the Envelope format, except that it does not include the `chatId`, `isEdited`, and `box` fields.

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

5.5. Retrieving the list of chats

Request:

```json
{
    "type": 1,
    "id": 1,
    "method": "getChats",
    "payload": {
        "count": 10,
        "page": 1
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Type of message (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	

Parameter	Type	Req.	Description
			Unique request identifier. This is an incremented value assigned by the sender, required in each request to associate with a response. Read more here
method	*string*	Yes	Command `getChats`
count	*uint32*	Yes	Maximum number of chat objects in the response
page	*uint32*	Yes	Page number of the request (paging). Starts at 1

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": [
        {
            "chatId": "2b896a6ef210d6b2dcaebc2186c9a7974d616054",
            "title": "TrueConf Server Administrators' Chat",
            "chatType": 2,
            "unreadMessages": 0,
            "lastMessage": {
                "messageId": "267d61f2-2ba1-4e88-83ab-bb84415f31b6",
                "timestamp": 1746028006345,
                "author": {
                    "id": "brown@video.example.com",
                    "type": 1
                },
                "type": 200,
                "content": {
                    "text": "What's up?",
                    "parseMode": "html"
                }
            }
        },
        {
            "chatId": "4a1f88f1070d2f43d385cde9ff61964bc6b74477",
            "title": "chester@video.example.com",
            "chatType": 1,
            "unreadMessages": 1,
            "lastMessage": {
                "messageId": "d4f756a2-8dd1-4cce-af7c-5a4311af0c79",
                "timestamp": 1746028006690,
```

```
                "author": {
                    "id": "chester@video.example.com",
                    "type": 1
                },
                "type": 200,
                "content": {
                    "text": "Hello!",
                    "parseMode": "html"
                }
            }
        },
        {
            "chatId": "5abba630dd1089090ba41c69c7aa34e4ba277a43",
            "title": "Project Announcements",
            "chatType": 6,
            "unreadMessages": 1,
            "lastMessage": {
                "messageId": "267d61f2-2ba1-4e88-83ab-bb84415f31b6",
                "timestamp": 1746028006345,
                "author": {
                    "id": "brown@video.example.com",
                    "type": 1
                },
                "type": 200,
                "content": {
                    "text": "Hello everyone!",
                    "parseMode": "html"
                }
            }
        }
    ]
}
```


Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used for linking the request and response
payload	*Array<Chat>*	Yes	User's chat array

> *i* In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

5.6. Deleting chat

Deleting both personal and group chats.

Request:

```json
{
  "type": 1,
  "id": 1,
  "method": "removeChat",
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256"
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incremented value assigned by the sender, required in each request for subsequent linkage with the response. Read more here
method	*string*	Yes	Command `removeChat`
chatId	*string*	Yes	Identifier of the chat to be removed

Answer:

```json
{
  "type": 2,
  "id": 1,
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256"
  }
}
```


Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to link the request and response
chatId	*string*	Yes	Remote chat identifier

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

5.7. Adding a participant to the chat

Request:

```json
{
  "type": 1,
  "id": 1,
  "method": "addChatParticipant",
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
    "userId": "user@video.example.com"
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required in every request for subsequent matching with a response. For more details, read here
method	*string*	Yes	The command `addChatParticipant`
chatId	*string*	Yes	The identifier of the chat to which the user is being added
userId	*string*	Yes	The TrueConf ID of the user being added to the chat

Answer:


```json
{
  "type": 2,
  "id": 1,
  "payload": {}
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Type of message (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to link the request and response
payload	*object*	Yes	In case of success, you will receive an empty object

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

5.8. Removing a participant from the chat

Request:

```json
{
  "type": 1,
  "id": 1,
  "method": "removeChatParticipant",
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
    "userId": "user@video.example.com"
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, mandatory in each request for subsequent linking to the response. Read more here

Parameter	Type	Required	Description
method	*string*	Yes	Command `removeChatParticipant`
chatId	*string*	Yes	Chat identifier from which the user is removed
userId	*string*	Yes	TrueConf ID of the user being removed

Answer:

```json
{
  "type": 2,
  "id": 1,
  "payload": {}
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Type of message (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to link the request and response
payload	*object*	Yes	In case of success, you will receive an empty object

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

5.9. Checking Participant Presence in Chat

Request:

```json
{
  "type": 1,
  "id": 1,
  "method": "hasChatParticipant",
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
    "userId": "user@video.example.com"
  }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementable value assigned by the sending party, required in every request for subsequent mapping with the response. Read more here
method	*string*	Yes	Command `hasChatParticipant`
chatId	*string*	Yes	The chat ID in which the user's presence is checked
userId	*string*	Yes	TrueConf ID of the user being checked

Answer:

```json
{
  "type": 2,
  "id": 1,
  "payload": {
    "result": true
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to associate the request and response
result	*boolean*	Yes	Flag indicating participant presence in the chat. If `true` — the participant is present in the chat

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

5.10. Retrieving the list of chat participants

Request:


```json
{
  "type": 1,
  "id": 1,
  "method": "getChatParticipants",
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
    "pageSize": 100,
    "pageNumber": 1
  }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Type of message (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required for every request to link with a response. More details read here
method	*string*	Yes	Command `getChatParticipants`
chatId	*string*	Yes	Chat identifier
pageSize	*uint32*	Yes	Number of records with chat participant information in the response to the request
pageNumber	*uint32*	Yes	Offset at which the list of participants will be returned. Calculated by the formula `startOffset = pageSize * pageNumber`

Answer:

```json
{
  "type": 2,
  "id": 1,
  "payload": {
    "participants": [
      {
        "userId": "user@video.example.com",
        "role": "user"
      },
      {
        "userId": "admin@video.example.com",
        "role": "admin"
```

```
        }
      ]
    }
  }
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to link the request and response
participants	*Array<ChatParticipant>*	Yes	A list of chat participants represented as ChatParticipant objects

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

6. Contacts Management

6.1. Retrieving the display name of a user by their TrueConf ID

Request:

```json
{
    "type": 1,
    "id": 1,
    "method": "getUserDisplayName",
    "payload": {
        "userId": "user@video.example.com"
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Type of message (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sending party, required in each request for subsequent linkage with the response. Read more here
method	*string*	Yes	Command `getUserDisplayName`
userId	*string*	Yes	User's TrueConf ID for which the name is to be retrieved

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": {
        "displayName": "Abe Chester"
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to link the request and response

Parameter	Type	Req.	Description
displayName	*string*	Yes	User's display name on the server

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

7. Working with messages

7.1. Sending a text message in chat

Request:

```json
{
  "type": 1,
  "id": 1,
  "method": "sendMessage",
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
    "content": {
      "text": "Hello, world!",
      "parseMode": "text"
    }
  }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incremented value assigned by the sender, required in each request for subsequent binding with the response. Read more here
method	*string*	Yes	Command `sendMessage`
chatId	*string*	Yes	Chat identifier
replyMessageId	*string*	No	If replying to a message, the field should contain the identifier of that message (see example below)
text	*string*	Yes	Message text
parseMode	*string*	Yes	Formatting mode (see below)

Answer:

```json
{
  "type": 2,
  "id": 1,
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
```

```
    "messageId": "8fe61d87-6db0-4792-8f5e-e5960c7d5a06",
    "timestamp": 1735314170572
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used for linking the request and response
chatId	*string*	Yes	Identifier of the chat to which the message was sent
messageId	*string*	Yes	Message identifier. This identifier can be used later for editing, forwarding, or deleting the message
timestamp	*uint64*	Yes	Message sending timestamp in UNIX timestamp format with millisecond precision

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

7.2. Message formatting

The chat bot API supports three message formatting types:

- `text` – plain text. All characters will be preserved and displayed.
- `markdown` - basic Markdown formatting is supported. Special characters are converted into formatted text (bold, italic, links, etc.);
- `html` - HTML tags will be interpreted and displayed as formatted text.

7.2.1. Markdown

Supported Markdown syntax:

Style	Syntax	Example	Displayed Text
Bold	`**` or `__`	`**bold text**`	**bold text**
Italic	`*` or `_`	`_italic text_`	*italic text*
Strikethrough	`~~`	`~~strikethrough text~~`	~~strikethrough text~~
Link			Website

Style	Syntax	Example	Displayed Text
	`[Text](https://url)`	`[Website](https://example.com)`	

7.2.2. HTML

Supported HTML markup syntax:

Tag	HTML Example	Displayed Text
`<b>`	`<b>bold text</b>`	**bold text**
`<i>`	`<i>italic text</i>`	*italic text*
`<s>`	`<s>strikethrough text</s>`	~~strikethrough text~~
`<u>`	`<u>underlined text</u>`	underlined text
`<a>`	`<a href="https://example.com">Website</a>`	Website

All additional tag attributes will be removed. For example, the HTML code:

```html
<i style="position: fixed; top:0">Hacked!</i>
```

will be converted to:

```html
<i>Hacked!</i>
```

7.3. Reply to an existing message

To reply to an existing message, you need to fill in the `replyMessageId` field with the identifier of the message you are responding to:

```json
{
  "type": 1,
  "id": 1,
  "method": "sendMessage",
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
    "replyMessageId": "3d968a21-543c-4fb2-9d8e-e31e3fbb35eb",
    "content": {
      "text": "Hello, world!",
      "parseMode": "text"
    }
```

```
  }
}
```

7.4. Editing an existing message

Request:

```json
{
  "type": 1,
  "id": 1,
  "method": "editMessage",
  "payload": {
    "messageId": "8fe61d87-6db0-4792-8f5e-e5960c7d5a06",
    "content": {
      "text": "Hello, world (edited)!",
      "parseMode": "text"
    }
  }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. It is an incrementing value assigned by the sender and is mandatory in every request for subsequent response association. Read more here
method	*string*	Yes	Command `editMessage`
messageId	*string*	Yes	Text message identifier
text	*string*	Yes	Message text
parseMode	*string*	Yes	Formatting mode (see above)

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": {
        "messageId": "8fe61d87-6db0-4792-8f5e-e5960c7d5a06",
        "timestamp": 1735314170572
```

```
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to link the request and response
timestamp	*uint64*	Yes	Message edit timestamp in UNIX timestamp format with millisecond precision
messageId	*string*	Yes	Identifier of the edited message

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

7.5. Forwarding a message to another chat

Request:

```json
{
    "type": 1,
    "id": 1,
    "method": "forwardMessage",
    "payload": {
        "chatId": "4a1f88f1070d2f43d385cde9ff61964bc6b74477",
        "messageId": "8fe61d87-6db0-4792-8f5e-e5960c7d5a06"
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incremented value assigned by the sender, required in each request to subsequently correlate with the response. More details read here
method	*string*	Yes	Command `forwardMessage`

Parameter	Type	Req.	Description
chatId	*string*	Yes	Identifier of the chat where you will forward the message
messageId	*string*	Yes	Identifier of the forwarded message

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": {
        "chatId": "4a1f88f1070d2f43d385cde9ff61964bc6b74477",
        "messageId": "3e168850-d9ea-45ec-a0f7-21c43e8ba2a8",
        "timestamp": 1735314170572
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to link the request and response
chatId	*string*	Yes	Unique identifier of the chat to which we forwarded the message
messageId	*string*	Yes	Unique identifier of the **new** message containing your forwarded message
timestamp	*uint64*	Yes	Timestamp of the message forwarding in UNIX timestamp format with millisecond precision

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

7.6. Retrieving a message by its ID

Request:

```json
{
    "type": 1,
```

```
    "id": 1,
    "method": "getMessageById",
    "payload": {
        "messageId": "8fe61d87-6db0-4792-8f5e-e5960c7d5a06"
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Type of message (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique identifier for the request. An incremented value assigned by the sender, required for each request to link with the response. Read more here
method	*string*	Yes	Command `getMessageById`
messageId	*string*	Yes	Identifier of the requested message

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": {
        "chatId": "2b896a6ef210d6b2dcaebc2186c9a7974d616054",
        "messageId": "8fe61d87-6db0-4792-8f5e-e5960c7d5a06",
        "timestamp": 1234567890,
        "author": {
            "id": "user@video.example.com",
            "type": 1
        },
        "replyMessageId": null,
        "isEdited": true,
        "box": {
            "id": 123456,
            "position": "0"
        },
        "type": 0,
        "content": {
            "text": "Hello, chat bot!",
            "parseMode": "html"
        }
    }
```

```
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Type of message (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to link the request and response
chatId	*string*	Yes	Unique identifier of the chat where the message is located
messageId	*string*	Yes	Unique identifier of the message
timestamp	*uint64*	Yes	Message send timestamp in UNIX timestamp format with millisecond precision
replyMessageId	*string*	No	If this is a reply to a message, contains the ID of that message
isEdited	*boolean*	Yes	Indicates whether the message has ever been edited
payload.type	*uint32*	Yes	Type of the nested message. See more at EnvelopeTypeEnum
author.id	*string*	Yes	Unique identifier of the message author. This is either the TrueConf ID of the user or the server name from which the message came. The type of identifier depends on the `type` field
author.type	*uint32*	Yes	Type of the message author. See more at EnvelopeAuthorTypeEnum
box.id	*number*	Yes	Sequential number of the "box" containing the message. See more in the relevant section
box.position	*string*	Yes	Sequential number of the message in the "box". See more in the relevant section

Depending on the value of `payload.type`, the `payload.content` object may contain data in one of the following formats:

7.6.0.1. Text message

Used when the message contains only text information.

Parameter	Type	Required	Description
text	*string*	Yes	Message text
parseMode	*string*	Yes	Formatting mode (see above)

7.6.0.2. Attachment (file)

Used when the message contains a file (document, image, etc.).

Parameter	Type	Required	Description
name	*string*	Yes	File name
size	*number*	Yes	File size
mimeType	*string*	Yes	File MIME type
fileId	*string*	Yes	File identifier on the server

7.6.0.3. Forwarded message

Represents an embedded message containing one of the structures described above (`text` , `attachment` , etc.) along with additional fields containing metadata. Structurally similar to a regular message, it includes `author` , `timestamp` , `content` , and other fields.

> *i* In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

7.7. Retrieving chat history

Request:

```json
{
    "type": 1,
    "id": 1,
    "method": "getChatHistory",
    "payload": {
        "chatId": "4a1f88f1070d2f43d385cde9ff61964bc6b74477",
        "count": 9999,
        "fromMessageId": "e37cfc52-592f-453f-a3dd-275170b2018d"
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required in each request to link it with a response. More details here
method	*string*	Yes	Command `getChatHistory`
chatId	*string*	Yes	Chat identifier for which the message history is requested
count	*uint32*	Yes	Number of messages to be returned in the response
fromMessageId	*string*	No	Chat message identifier from which the history will be returned

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": {
        "chatId": "4a1f88f1070d2f43d385cde9ff61964bc6b74477",
        "count": 1,
        "messages": [
            {
                "messageId": "8fe61d87-6db0-4792-8f5e-e5960c7d5a06",
                "timestamp": 1234567890,
                "author": {
                    "id": "user@video.example.com",
                    "type": 1
                },
                "replyMessageId": null,
                "isEdited": true,
                "box": {
                    "id": 123456,
                    "position": ""
                },
                "type": 0,
                "content": {
                    "text": "Hello, chat bot!",
                    "parseMode": "html"
                }
```

```
            }
        ]
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to link the request and response
chatId	*string*	Yes	Chat ID for which the message history was requested
count	*uint32*	Yes	Number of messages received
messages	*Array<Envelope>*	No	An array of `Envelope` objects, each representing a single message. See parameter descriptions above

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

7.8. Deleting a message

Request:

```json
{
    "type": 1,
    "id": 1,
    "method": "removeMessage",
    "payload": {
        "messageId": "8fe61d87-6db0-4792-8f5e-e5960c7d5a06",
        "forAll": true
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	

Parameter	Type	Req.	Description
			Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required in each request for subsequent binding with the response. Read more here
method	*string*	Yes	Command `removeMessage`
messageId	*string*	Yes	Identifier of the message to be removed
forAll	*boolean*	Yes	If `true`, the message will be removed for everyone

Answer:

```json
{
  "type": 2,
  "id": 1,
  "payload": {}
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Type of message (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to link the request and response
payload	*object*	Yes	In case of success, you will receive an empty object

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

8. Working with files

8.1. File Transfer

File transfer is carried out in three stages:

1. **Creating a File Upload Task**. The client initiates the upload creation and receives the necessary identifiers.
2. **Uploading a file to the server's file storage**. The file is uploaded to the server using an HTTP request (see the example below).
3. **Sending a message with an attached file**. A message is created in the chat specifying the identifier of a previously uploaded file.

8.1.1. Creating a file upload task

Request:

```json
{
  "type": 1,
  "id": 5,
  "method": "uploadFile",
  "payload": {
    "fileSize": 1487
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sending party, required in every request for subsequent matching with the response. Read more here
method	*string*	Yes	Command `uploadFile`
fileSize	*number*	Yes	File size in bytes

Answer:

```json
{
  "type": 2,
  "id": 5,
  "payload": {
    "uploadTaskId": "2e9537c4-e82c-467c-a149-3c1b41326def"
```

```
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to link the request and response
uploadTaskId	*string*	Yes	File upload task identifier

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

8.1.2. Uploading a file to the file storage

To upload a file to the storage, you need to make an HTTP request to the server. The parameters listed below are specified in the **HTTP headers**, not in the URL (*query string*):

```
POST /bridge/api/client/v1/files HTTP/1.1
Upload-Task-Id: 2e9537c4-e82c-467c-a149-3c1b41326def
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundaryQeFqWQPxCmHvCIyc
Content-Length: 2081

------WebKitFormBoundaryQeFqWQPxCmHvCIyc
Content-Disposition: form-data; name="file";
filename="some_image.png"
Content-Type: image/png

<file binary data>
------WebKitFormBoundaryQeFqWQPxCmHvCIyc
Content-Disposition: form-data; name="preview"; filename="d970f81f-
d433-401c-8182-d33c59b96cd5"
Content-Type: image/webp

<preview binary data>
------WebKitFormBoundaryQeFqWQPxCmHvCIyc
```

Title	Req.	Description
Upload-Task-Id	Yes	The file upload task identifier previously obtained from the uploadFile command

The request body must be presented as a `FormData` object with the type `multipart/form-data` and must contain the required field `file`, which holds the file itself. If the file has a preview, it should be included in the form under the `preview` field, and the name under which the preview is placed must differ from the name of the file itself. Both the file and the preview are included in binary format.

Answer:

If the upload is successful, the HTTP server will respond with the code `200 OK`:

```
HTTP/1.1 200 OK
Allow: OPTIONS, GET, HEAD, POST
Server: TrueConf Bridge
Content-Type: application/json
Content-Length: 57
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: height,mime,request_id,preview,cid

{
  "temporalFileId": "2e9537c4-e82c-467c-a149-3c1b41326def"
}
```

The response body contains the `temporalFileId` field, which is a temporary file identifier ready for sending in a message. After sending, this identifier loses any useful properties.

Error:

In case of an error, the HTTP server will respond with the code `400 Bad Request`:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 62

{
  "error": 310,
  "error_description": "No upload task ID provided",
}
```

8.1.3. Sending a file in a message

Request:

```json
{
  "type": 1,
  "id": 6,
  "method": "sendFile",
  "payload": {
    "chatId": "bea15543d971a83c8ba1f103104791762c6b4b8f",
    "content": {
      "temporalFileId": "32506b12-6a1a-4cd3-be24-373c7bcf9510"
    }
  }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required in each request for subsequent linking with the response. Read more here
method	*string*	Yes	Command `sendFile`
chatId	*string*	Yes	Chat identifier for sending the message
temporalFileId	*string*	Yes	Temporary file identifier received in response to an HTTP request

Answer:

```json
{
  "type": 2,
  "id": 6,
  "payload": {
    "chatId": "bea15543d971a83c8ba1f103104791762c6b4b8f",
    "timestamp": 1735134222098,
    "messageId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
    "fileId": "e09dbd63a16bd8cd7bddda7193c1c4ff0d2f6de5"
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to correlate request and response
chatId	*string*	Yes	Chat identifier
timestamp	*number*	Yes	Message sending timestamp in UNIX timestamp format with millisecond precision
messageId	*string*	Yes	Identifier of the sent message
fileId	*string*	Yes	Permanent identifier of the sent file for retrieving information about it

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

8.2. Retrieving file information and downloading the file

To obtain detailed information about an incoming file in the chat, you need to use the `payload.content.id` value obtained from the incoming new message request.

Request:

```json
{
  "method": "getFileInfo",
  "id": 9,
  "type": 1,
  "payload": {
    "fileId": "e09dbd63a16bd8cd7bddda7193c1c4ff0d2f6de5"
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	

Parameter	Type	Required	Description
			Unique request identifier. An incremented value assigned by the sender, required in each request to link with the response. Read more here
method	*string*	Yes	The command `getFileInfo`
fileId	*string*	Yes	File identifier from the incoming new message request

Answer:

```json
{
    "type": 2,
    "id": 9,
    "payload": {
        "name": "AirHelp_Logo.webp",
        "size": 3446,
        "mimeType": "image/webp",
        "downloadUrl": "https://10.111.15.116/bridge/api/client/v1/
files?file_id=e09dbd63a16bd8cd7bddda7193c1c4ff0d2f6de5",
        "readyState": 2,
        "previews": [
            {
                "name": "preview-2278f47b821f72e4a44ab6b980180518.we
bp",
                "mimeType": "image/webp",
                "size": 1264,
                "downloadUrl": "https://10.111.15.116/bridge/api/
client/v1/files/preview?
file_id=e09dbd63a16bd8cd7bddda7193c1c4ff0d2f6de5"
            }
        ],
        "infoHash": "e09dbd63a16bd8cd7bddda7193c1c4ff0d2f6de5"
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to link the request and response

Parameter	Type	Req.	Description
name	*string*	Yes	File name
size	*number*	Yes	File size in bytes
mimeType	*string*	Yes	MIME type of the file
downloadUrl	*string*	Yes	URL for downloading the file
readyState	*FileReadyStateEnum*	Yes	File state on the server
infoHash	*string*	Yes	File identifier on the server
previews	*Array*	Yes	Array of file previews, repeating the parameters described above

* To download a file or its preview, you need to send an HTTP GET request to the address specified in `downloadUrl`.

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

8.3. Subscription to file upload progress on the server

If the file is in the `UPLOADING` status, you can subscribe to the upload process to be notified when the file becomes available.

Request:

```json
{
  "method": "subscribeFileProgress",
  "id": 10,
  "type": 1,
  "payload": {
    "fileId": "e09dbd63a16bd8cd7bddda7193c1c4ff0d2f6de5"
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST

Parameter	Type	Required	Description
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required in each request to link with the response. Read more here
method	*string*	Yes	Command `subscribeFileProgress`
fileId	*string*	Yes	File identifier on the server

Answer:

```json
{
  "id": 10,
  "type": 2,
  "payload": {
    "result": true
  }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to correlate the request and response
result	*boolean*	Yes	Indicates whether the subscription to the upload was successful

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

8.4. Receiving file upload events

After subscribing to the upload, `uploadingProgress` events will start arriving via WebSocket:

```json
{
  "id": 3,
  "type": 1,
  "method": "uploadingProgress",
```

```
  "payload": {
    "fileId": "e09dbd63a16bd8cd7bddda7193c1c4ff0d2f6de5",
    "progress": 123
  }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Type of message (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST from the server
id	*uint32*	Yes	Identifier matching the number sent in the original request, used for linking the request and response
fileId	*string*	Yes	File identifier on the server
progress	*number*	Yes	Number of bytes uploaded to the server

When a message is received where the value of `progress` equals the value of the `size` field from the getFileInfo response, the file is considered fully uploaded and can be retrieved using a GET request. After this, no more events with updated progress will be sent. If the file was already uploaded earlier, only one final event equivalent to the last one will be sent.

8.5. Unsubscribe from receiving upload event notifications

If necessary, you can unsubscribe from receiving file upload events on the server.

Request:

```json
{
  "method": "unsubscribeFileProgress",
  "id": 10,
  "type": 1,
  "payload": {
    "fileId": "e09dbd63a16bd8cd7bddda7193c1c4ff0d2f6de5"
  }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required in each request to link it with the response. For more details, read here

Parameter	Type	Req.	Description
method	*string*	Yes	Command `unsubscribeFileProgress`
fileId	*string*	Yes	File identifier on the server

Answer:

```json
{
  "id": 10,
  "type": 2,
  "payload": {
    "result": true
  }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier matching the number sent in the original request, used to correlate the request and response
result	*boolean*	Yes	Whether the unsubscription from file upload events was successful

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

9. Working with Polls

9.1. Sending a poll message in chat

Request:

```json
{
    "type": 1,
    "id": 5,
    "method": "sendSurvey",
    "payload": {
        "chatId": "5abba630dd1089090ba41c69c7aa34e4ba277a43",
        "replyMessageId": "267d61f2-2ba1-4e88-83ab-bb84415f31b6",
        "content": {
            "url": "https://server.url/webtools/survey",
            "appVersion": 1,
            "path": "employee_testing",
            "title": "Employee survey",
            "description": "{{Survey}}",
            "buttonText": "{{Go to survey}}",
            "secret": "25690753a489f037af09b5cbce417b41374807fe",
            "alt": "📊 <a href='https://server.url/webtools/survey?id=employee_testing&error=autologin_not_supported'>Employee survey</a>"
        }
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Type of message (default `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sending side, mandatory in each request for subsequent linking with the response. More details read here.
method	*string*	Yes	Command `sendSurvey`
chatId	*string*	Yes	Chat identifier for sending the survey
replyMessageId	*string*	No	Message identifier if this is a reply

Parameter	Type	Required	Description
url	*string*	Yes	String URL for surveys on the server, formatted as `https://server.name/webtools/survey`
appVersion	*number*	Yes	Version of the surveys on the server
path	*string*	Yes	Survey campaign identifier
title	*string*	Yes	Survey title
description	*string*	Yes	Type of survey — anonymous or non-anonymous. Always contains the string `\{\{Anonymous survey\}\}` or `\{\{Survey\}\}`
buttonText	*string*	Yes	Text displayed on the button to proceed to the survey. Always contains the string `\{\{Go to survey\}\}`
secret	*string*	Yes	Random string from the message body. Must be generated as a SHA1 hash from the concatenation of the survey title and an arbitrary string.
alt	*string*	Yes	Alternative link to the survey. Formatted as: `📊 <a href="<url>?id=<path>"><title></a>`, where the respective values are substituted for `<url>`, `<path>`, and `<title>`.

Answer:

```json
{
    "type": 2,
    "id": 5,
    "payload": {
        "timestamp": 1750168761373,
        "messageId": "f8865b5c-877c-4a55-b175-a667ebf5f007",
        "chatId": "5abba630dd1089090ba41c69c7aa34e4ba277a43"
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE

Parameter	Type	Required	Description
id	*uint32*	Yes	Identifier matching the number sent in the original request, used to link request and response
chatId	*string*	Yes	Identifier of the chat where the message was sent
messageId	*string*	Yes	Message identifier. This identifier can be used for further modification, forwarding, or deletion of the message
timestamp	*uint64*	Yes	Message sending timestamp in UNIX timestamp format with millisecond precision

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

9.2. Editing a poll message

Request:

```json
{
    "type": 1,
    "id": 1,
    "method": "editSurvey",
    "payload": {
        "messageId": "f8865b5c-877c-4a55-b175-a667ebf5f007",
        "content": {
            "path": "employee_testing_marketing",
            "title": "Employee survey of the Marketing department",
            "description": "{{Survey}}",
            "buttonText": "{{Go to survey}}",
            "alt": "📊 <a href='https://server.url/webtools/survey?id=employee_testing&error=autologin_not_supported'><title></a>"
        }
    }
}
```

Parameter	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST

Parameter	Type	Req.	Description
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, required in each request for subsequent binding with the response. Read more here
method	*string*	Yes	Command `editSurvey`
messageId	*string*	Yes	Survey message identifier
path	*string*	Yes	Survey campaign identifier
title	*string*	Yes	Survey title
description	*string*	Yes	Survey type — anonymous or non-anonymous. Always contains the string `\{\{Anonymous survey\}\}` or `\{\{Survey\}\}`
buttonText	*string*	Yes	Text to be displayed on the button to proceed to the survey. Always contains the string `\{\{Go to survey\}\}`
alt	*string*	Yes	Alternative link to proceed to the survey. It appears as: 📊 `<a href="<url>?id=<path>"><title></a>` , where `<url>` , `<path>` , and `<title>` are replaced with the corresponding values.

Answer:

```json
{
    "type": 2,
    "id": 1,
    "payload": {
        "messageId": "f8865b5c-877c-4a55-b175-a667ebf5f007",
        "timestamp": 1735314170572
    }
}
```

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	An identifier that matches the number sent in the original request, used for linking the request and response
messageId	*string*	Yes	Identifier of the edited message
timestamp	*uint64*	Yes	Timestamp of the message edit in UNIX timestamp format with millisecond precision

i In case of an error, a message containing the `errorCode` parameter is returned. A list of possible values is available in the relevant section of the documentation.

10. Server Requests (Server Messages)

This section describes messages that **the server sends on its own initiative**, without a direct client request. These are used for notifications about external events, state changes, and actions of other users. For example:

- **createGroupChat** – group chat created.
- **addChatParticipant** – user added to group chat.
- **sendMessage** – new chat message.

For each server request, the client must respond as follows:

```json
{
    "type": 2,
    "id": 123456
}
```

10.1. Personal chat created

Message from the server:

```json
{
    "method": "createP2PChat",
    "type": 1,
    "id": 2,
    "payload": {
        "chatId": "bd05af54347e04a1c44e70033d35834d4428bb5d",
        "title": "brown@video.example.com",
        "chatType": 1,
        "lastMessage": {
            "chatId": "bd05af54347e04a1c44e70033d35834d4428bb5d",
            "messageId": "d8bdd362-2656-46ff-a1db-8dadd2b76250",
            "timestamp": 1746028006345,
            "author": {
                "id": "brown@video.example.com",
                "type": 1
            },
            "type": 200,
            "content": {
                "text": "What's up?",
                "parseMode": "text"
            }
        },
        "unreadMessages": 1
```

```
    }
}
```

Parameter	Type	Required	Description
method	*string*	Yes	Command `createP2PChat`
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender. Read more here.
chatId	*string*	Yes	Identifier of the created chat
title	*string*	Yes	TrueConf ID of the user who created the chat
chatType	*uint32*	Yes	Type of the created chat. Corresponds to ChatTypeEnum
lastMessage.chatId	*string*	Yes	Chat identifier where the message is located
messageId	*string*	Yes	Message identifier. Can be used later for editing, forwarding, or deleting the message
timestamp	*uint64*	Yes	Message sending timestamp in UNIX timestamp format with millisecond precision
author.id	*string*	Yes	Unique identifier of the message author (TrueConf ID or server name)
author.type	*uint32*	Yes	Type of message author. See EnvelopeAuthorTypeEnum
lastMessage.type	*uint32*	Yes	Type of message, corresponds to EnvelopeTypeEnum
text	*string*	No	Message text
parseMode	*string*	No	Formatting mode (more details)
unreadMessages	*uint32*	Yes	Number of unread messages in the personal chat

i The `payload.lastMessage.content` field contains different data depending on the `payload.lastMessage.type` message type.

Response for the server:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

10.2. Group chat created.

Message from the server:

```json
{
    "method": "createGroupChat",
    "type": 1,
    "id": 14,
    "payload": {
        "chatId": "08d5dbda94a9de4b7554e3b4355307f9e97ffdb7",
        "title": "Marketing",
        "chatType": 2,
        "lastMessage": {
            "chatId": "08d5dbda94a9de4b7554e3b4355307f9e97ffdb7",
            "messageId": "cccea4a7-24b4-4b2c-8c50-67d01bce17bf",
            "timestamp": 1746029638147,
            "author": {
                "id": "user@video.example.com",
                "type": 0
            },
            "type": 110,
            "content": {
                "userId": "bot@video.example.com",
                "role": "user"
            }
        },
        "unreadMessages": 2
    }
}
```

Parameter	Type	Req.	Description
method	*string*	Yes	Command `createGroupChat`
type	*uint32*	Yes	Message type (default `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incremented value assigned by the sender, required in each request for subsequent response matching. Read more here
chatId	*string*	Yes	Identifier of the created chat
title	*string*	Yes	Title of the created chat
chatType	*uint32*	Yes	Type of the created chat. Corresponds to ChatTypeEnum
lastMessage.chatId	*string*	Yes	Identifier of the chat containing the message
messageId	*string*	Yes	Message identifier. Can be used for modification, forwarding or deletion
timestamp	*uint64*	Yes	Timestamp in UNIX timestamp format with millisecond precision
author.id	*string*	Yes	TrueConf ID of the message author or server name
author.type	*uint32*	Yes	Type of author. See EnvelopeAuthorTypeEnum
lastMessage.type	*uint32*	Yes	Message type. See EnvelopeTypeEnum
userId	*string*	No	TrueConf ID of the user who sent the message
role	*string*	No	User's role in the chat (see more)
unreadMessages	*uint32*	Yes	Number of unread messages in the group chat

i The `payload.lastMessage.content` field varies depending on the `payload.lastMessage.type` of the message.

Response from client:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

10.3. Channel created

Message from the server:

```json
{
    "method": "createChannel",
    "type": 1,
    "id": 14,
    "payload": {
        "chatId": "08d5dbda94a9de4b7554e3b4355307f9e97ffdb7",
        "title": "Important Announcements",
        "chatType": 6,
        "lastMessage": {
            "chatId": "08d5dbda94a9de4b7554e3b4355307f9e97ffdb7",
            "messageId": "cccea4a7-24b4-4b2c-8c50-67d01bce17bf",
            "timestamp": 1746029638147,
            "author": {
                "id": "user@video.example.com",
                "type": 0
            },
            "type": 110,
            "content": {
                "userId": "bot@video.example.com",
                "role": "user"
            }
        },
        "unreadMessages": 2
    }
}
```

Parameter	Type	Req.	Description
method	*string*	Yes	Command `createChannel`
type	*uint32*	Yes	Message type (default `1`). Corresponds to MESSAGE_TYPE.REQUEST

Parameter	Type	Req.	Description
id	*uint32*	Yes	Unique request identifier. An incremented value assigned by the sender, required in each request for subsequent matching with the response. More details read here
chatId	*string*	Yes	Identifier of the created channel
title	*string*	Yes	Name of the created channel
chatType	*uint32*	Yes	Type of the created channel. Corresponds to ChatTypeEnum = 6
lastMessage.chatId	*string*	Yes	Identifier of the channel where the message is located
messageId	*string*	Yes	Identifier of the message. This identifier can be used later for editing, forwarding, or deleting the message
timestamp	*uint64*	Yes	Timestamp of the message sent in UNIX timestamp format with millisecond precision
author.id	*string*	Yes	Unique identifier of the message author. It is either the TrueConf ID of the user or the server name from which the message was sent. The type of identifier depends on the `type` field
author.type	*uint32*	Yes	Type of the message author. For more details, see EnvelopeAuthorTypeEnum
lastMessage.type	*uint32*	Yes	Type of message. Allows identification of the type of content in the message. Corresponds to EnvelopeTypeEnum
userId	*string*	No	TrueConf ID of the user who sent the message
role	*string*	No	User role in the group chat (see more details)
unreadMessages	*uint32*	Yes	Number of unread messages in the channel

i The `payload.lastMessage.content` field varies depending on the `payload.lastMessage.type` of the message.

Response from client:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

10.4. Participant added to chat

Message from the server:

```json
{
    "method": "addChatParticipant",
    "type": 1,
    "id": 1,
    "payload": {
        "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
        "userId": "user@video.example.com",
        "addedBy": {
            "id": "admin@video.example.com",
            "type": 1
        },
        "timestamp": "1735370776"
    }
}
```

Parameter	Type	Required	Description
method	*string*	Yes	Command `addChatParticipant`
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementable value assigned by the sender. More details read here
chatId	*string*	Yes	Identifier of the chat where the user was added
userId	*string*	Yes	TrueConf ID of the user added to the chat
addedBy.id	*string*	Yes	Unique identifier of the user who added the new participant. This is either the TrueConf ID or the server name, depending on `addedBy.type`
addedBy.type	*uint32*	Yes	Type of the user who added the new participant. See EnvelopeAuthorTypeEnum
timestamp	*uint64*	Yes	

Parameter	Type	Required	Description
			Timestamp of adding the participant to the chat in UNIX timestamp format (in milliseconds)

Response from client:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

10.5. Chat participant removed

Message from the server:

```json
{
  "method": "removeChatParticipant",
  "type": 1,
  "id": 1,
  "payload": {
    "chatId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
    "userId": "user@video.example.com",
    "removedBy": {
      "id": "admin@video.example.com",
      "type": 1
    },
    "timestamp": "1735370778"
  }
}
```

Parameter	Type	Required	Description
method	*string*	Yes	The `removeChatParticipant` command
type	*uint32*	Yes	Message type (default `1`). Corresponds to MESSAGE_TYPE.REQUEST

Parameter	Type	Required	Description
id	*uint32*	Yes	Unique request identifier. An incremental value assigned by the sender, required in each request to subsequently link it with the response. More details here
chatId	*string*	Yes	The chat identifier from which the user was removed
userId	*string*	Yes	The TrueConf ID of the user removed from the chat
removedBy.id	*string*	Yes	Unique identifier of the user who removed the participant from the chat. This is either the TrueConf ID or the server name from which the message was sent. The identifier type depends on the `removedBy.type` field
removedBy.type	*uint32*	Yes	Type of user who removed the participant from the chat. See more in EnvelopeAuthorTypeEnum
timestamp	*uint64*	Yes	Timestamp of the user's removal from the chat in UNIX timestamp format with millisecond precision

Response from client:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

10.6. New message in chat

Message from the server:

```json
{
    "method": "sendMessage",
    "type": 1,
    "id": 11,
    "payload": {
        "chatId": "bd05af54347e04a1c44e70033d35834d4428bb5d",
```

```
        "messageId": "d66254de-9d89-4130-8027-c5378f042800",
        "timestamp": 1746029007569,
        "author": {
            "id": "brown@video.example.com",
            "type": 1
        },
        "isEdited": false,
        "box": {
            "id": 4,
            "position": "0"
        },
        "type": 200,
        "content": {
            "text": "Text",
            "parseMode": "html"
        }
    }
}
```


* The notification of a new message includes fields corresponding to the Envelope object. Detailed descriptions of these fields can be found in the section on working with messages.

Response from client:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

10.7. New file in chat

In the TrueConf chat protocol, the file send event occurs immediately upon message creation, without waiting for the file to be uploaded to the server. After this, the file will upload for a while before becoming available for download.

Message from the server:

```json
{
    "type": 1,
    "id": 7,
    "method": "sendMessage",
    "payload": {
        "chatId": "bea15543d971a83c8ba1f103104791762c6b4b8f",
        "messageId": "c8c3eee8-9ad0-4638-9692-ad16391a4256",
        "timestamp": 1735302531000,
        "author": {
            "id": "user@video.example.com",
            "type": 1
        },
        "isEdited": false,
        "box": {
            "id": 10,
            "position": ""
        },
        "type": 2,
        "content": {
            "name": "someimage.png",
            "mimeType": "image/png",
            "size": 2156,
            "fileId": "e09dbd63a16bd8cd7bddda7193c1c4ff0d2f6de5"
        }
    }
}
```

* A notification about a message with a file contains fields corresponding to the Envelope and FileMessageContent objects. Detailed descriptions of these fields can be found in the section dedicated to message handling.

Response from client:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

10.8. New poll in chat

Message from the server:

```json
{
    "method": "sendMessage",
    "type": 1,
    "id": 11,
    "payload": {
        "chatId": "bd05af54347e04a1c44e70033d35834d4428bb5d",
        "messageId": "d66254de-9d89-4130-8027-c5378f042800",
        "timestamp": 1746029007569,
        "author": {
            "id": "brown@video.example.com",
            "type": 1
        },
        "isEdited": false,
        "box": {
            "id": 4,
            "position": "0"
        },
        "type": 204,
        "content": {
            "url": "<tool_url>?
id=...&k=...&mode=popup&call_id=vasya@srv.trueconf.name/
1vca3&lang=ru&version=1&app=TrueConf+WebClient&dn=Vasiliy&s=12fee560
62786c267cc286045f1fac76",
            "appVersion": 1,
            "path": "some_survey",
            "title": "Meeting survey",
            "description": "{{Survey}}",
            "buttonText": "{{Go to survey}}",
            "secret": "054c1cf18f1e64f4c38b256effedfe18debdcbba",
            "alt": "📊 <a href='https://server.url/webtools/survey?
id=some_survey&error=autologin_not_supported'>Meeting survey</a>"
        }
```

```
    }
}
```

* The notification about a message with a survey includes fields corresponding to the Envelope objects. A detailed description of these fields can be found in the section dedicated to working with messages and surveys.

Response from client:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

10.9. The message has been modified.

Message from the server:

```json
{
    "method": "editMessage",
    "type": 1,
    "id": 12,
    "payload": {
        "messageId": "d4d20b4f-e4a4-4abc-8e66-ff2e17f483b7",
        "timestamp": 1746029430000,
        "content": {
            "text": "🤝",
            "parseMode": "text"
        }
    }
}
```

Parameter	Type	Required	Description
method	*string*	Yes	Command `editMessage`

Parameter	Type	Required	Description
type	*uint32*	Yes	Message type (default `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. An incrementing value assigned by the sender, mandatory in each request to link with the response. More details read here
messageId	*string*	Yes	Identifier of the edited message. This identifier can be used later for copying, forwarding, or deleting the message
timestamp	*uint64*	Yes	Timestamp of the message modification in UNIX timestamp format with millisecond precision
text	*string*	Yes	Text of the edited message
parseMode	*string*	Yes	Formatting mode (see more details)

Response from client:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

10.10. The message has been deleted

Message from the server:

```json
{
    "method": "removeMessage",
    "type": 1,
    "id": 12,
    "payload": {
        "chatId": "bd05af54347e04a1c44e70033d35834d4428bb5d",
        "messageId": "d4d20b4f-e4a4-4abc-8e66-ff2e17f483b7",
        "removedBy": {
            "id": "brown@video.example.com",
```

```
            "type": 1
        }
    }
}
```

Parameter	Type	Required	Description
method	*string*	Yes	Command `removeMessage`
type	*uint32*	Yes	Message type (default is `1`). Corresponds to MESSAGE_TYPE.REQUEST
id	*uint32*	Yes	Unique request identifier. Incremental value assigned by the sender, required in each request for subsequent linking with the response. Read more here
chatId	*string*	Yes	Identifier of the chat where the message was removed
messageId	*string*	Yes	Identifier of the removed message. *This identifier can be used later for copying, forwarding, or deleting the message*
removedBy.id	*string*	Yes	Unique identifier of the user who removed the message. This is either the TrueConf ID or the name of the server from which the message originated. The type of identifier depends on the `removedBy.type` field
removedBy.type	*uint32*	Yes	Type of user who removed the message. For more details, see EnvelopeAuthorTypeEnum

Response from client:

```json
{
    "type": 2,
    "id": 123456
}
```

Field	Type	Req.	Description
type	*uint32*	Yes	Message type (default is `2`). Corresponds to MESSAGE_TYPE.RESPONSE
id	*uint32*	Yes	Identifier of the message being responded to (see details)

11. API Objects

This section presents the main data structures used for message exchange via the WebSocket protocol.

TrueConf Server API for Chat Bots is built on a messaging model where each entity is described in TypeScript using interfaces (`interface`) and enumerations (`enum`). This allows for a strict structure for both client requests and server responses and events.

- **Interfaces** define the format of messages and their payload.
- **Enums** describe the permissible values for message fields (e.g., message types, error codes, etc.).

These definitions form the basis for the serialization and processing of messages on the client and server.

11.1. Enums

11.1.1. ChatParticipantRoleEnum

The enumeration contains a list of possible participant roles in a chat:

```ts
enum ChatParticipantRoleEnum {
    OWNER = 0,
    ADMIN = 1,
    USER = 2,
    CONF_OWNER = 3,
    CONF_MODERATOR = 4,
    FAVORITES_OWNER = 5,
    WRITER = 6
}
```

where:

- `OWNER` - chat owner;
- `ADMIN` - designated chat administrator;
- `USER` - regular chat participant;
- `CONF_OWNER` - this role is only present in conference chats and indicates that the user is the conference owner;
- `CONF_MODERATOR` - this role is present only in conference chats and indicates that the user is a conference moderator;
- `FAVORITES_OWNER` - this role is only available in the "Favorites" chat (Saved messages).
- `WRITER` - this role is only present in the channel and indicates that the user can write messages in the channel.

You can review the description of the rights for each role in the dedicated section.

11.1.2. MESSAGE_TYPE

There are three types of messages:

```ts
enum MESSAGE_TYPE {
  RESERVED = 0,
  REQUEST = 1,
  RESPONSE = 2
}
```

Only REQUEST and RESPONSE are applicable.

11.1.3. OAUTH_ERROR

Error codes, according to the OAuth 2.0 specification, are presented as ASCII strings from the list specified in the specification. The following error codes are supported:

```ts
enum OAUTH_ERROR {
  INVALID_REQUEST = "invalid_request",
  INVALID_CLIENT = "invalid_client",
  INVALID_GRANT = "invalid_grant",
  UNSUPORTED_GRANT_TYPE = "unsupported_grant_type"
}
```

Error Code	Value	Description
`invalid_request`	Invalid Request	The request headers or body are improperly formatted. For instance, required fields are missing or the *Content-Type* header is incorrect.
`invalid_client`	Invalid Client	The value of the *client_id* field differs from the allowed value (e.g., it is not equal to `chat_bot`).
`invalid_grant`	Invalid Authorization Data	The user with the specified login is not found, or the password is incorrect.
`unsupported_grant_type`	Invalid Authorization Type	The value of the *grant_type* field differs from the allowed value (e.g., it is not equal to `password`).

11.1.4. ChatTypeEnum

The enumeration contains possible chat types.

```ts
enum ChatTypeEnum {
    UNDEF = 0,
    P2P = 1,
    GROUP = 2,
    SYSTEM = 3,
    FAVORITES = 5,
    CHANNEL = 6
}
```

where:

- `UNDEF` - unknown chat type
- `P2P` - personal messages (chat with another user)
- `GROUP` - group chat
- `SYSTEM` - system chat
- `FAVORITES` - "Favorites" chat (saved messages)
- `CHANNEL` - channel

11.1.5. EnvelopeTypeEnum

The enumeration contains the message type. Using this enumeration, you can determine the content type in the message.

```ts
enum EnvelopeTypeEnum {
    ADD_PARTICIPANT = 1,
    REMOVE_PARTICIPANT = 2,
    PLAIN_MESSAGE = 200,
    FORWARDED_MESSAGE = 201,
    ATTACHMENT = 202,
    SURVEY = 204,
}
```

where:

- `ADD_PARTICIPANT` – system message for adding a user to the chat.
- `REMOVE_PARTICIPANT` – system message for removing a user from the chat.
- `PLAIN_MESSAGE` – a regular text message from the server user.
- `FORWARDED_MESSAGE` – forwarded message.
- `ATTACHMENT` – a message containing information about an attachment (file).
- `SURVEY` - poll message

It should be noted that system messages, such as `ADD_PARTICIPANT` or `REMOVE_PARTICIPANT`, do not confirm the occurrence of an action, like a participant appearing in the chat. They are used solely for logging purposes. Separate commands and events exist for each action, for instance, adding a new participant to the chat.

To determine if a message is a system message, simply check that the value of the `type` parameter is less than 200.

11.1.6. EnvelopeAuthorTypeEnum

```ts
enum EnvelopeAuthorTypeEnum {
    SYSTEM,
    USER,
}
```

- `SYSTEM` - the message is authored by the server. In most cases, this indicates that the message is a system message. The server identifier (TrueConf Server Name) will be used as the author's identifier.
- `USER` – the author of the message is a server user. The TrueConf ID of this user will be used as the author's identifier.

11.1.7. TextParseModeEnum

This enumeration is used to specify the mode for processing the text content of messages.

```ts
enum TextParseModeEnum {
    TEXT = "text",
    MARKDOWN = "markdown",
    HTML = "html",
}
```

where:

- `TEXT` – the message content will be processed as text. All HTML tags will be escaped (converted to HTML entities).
- `MARKDOWN` - the content will be processed as markdown. The list of supported markdown rules is provided below.
- `HTML` – the content will be processed as HTML. The list of supported HTML tags is provided below.

11.1.8. FileReadyStateEnum

This enumeration is used to indicate the status of a file on the server.


```ts
enum FileReadyStateEnum {
    NOT_AVAILABLE = 0,
    UPLOADING = 1,
    READY = 2
}
```

where:

- `NOT_AVAILABLE` – the file has been removed due to quota restrictions, storage time, etc.;
- `UPLOADING` – the file is being uploaded;
- `READY` – file uploaded.

11.2. Structures (Interfaces)

11.2.1. AbstractEnvelopeContent

This object serves as a basic structure for storing message data. Currently, the object does not contain any specific fields, but they may be added in the future:

```ts
interface AbstractEnvelopeContent {}
```

11.2.2. AddParticipantEnvelopeContent

This object is used as the content of a message with the type EnvelopeTypeEnum.ADD_PARTICIPANT, which is sent when a participant is added to the chat, and has the following interface:

```ts
interface AddParticipantEnvelopeContent extends ChatParticipant,
AbstractEnvelopeContent {}
```

The object does not have its own fields and inherits fields from the ChatParticipant object.

Example:

```json
{
  "chatId": "51ffe1b1-1515-498e-8501-233116adf9da",
  "messageId": "2913538a-90e3-4323-9048-b599e4248796",
  "timestamp": 1735330912,
  "author": {
    "id": "trueconf.server.name",
    "type": 0 //0 - server, 1 - user
```

```
  },
  "replyMessageId": null, //string if this message is a reply
  "isEdited": false, //true if edited
  "box": {
    "id": 2,
    "position": "" //for sorting undelivered messages
  },
  "type": 1, //1 - ADD_PARTICIPANT
  "content": {
    "userId": "user@video.example.com",
    "role": "owner"
  }
}
```

11.2.3. AuthRequest

The authorization request message is as follows:

```ts
interface AuthRequest extends RequestMessage<"auth",
AuthRequestPayload> {}
```

The AuthRequest interface inherits fields from the AuthRequestPayload object, essentially serving as the message body, so the description of the fields will be provided there.

11.2.4. AuthRequestPayload

The message body (payload) for the authorization request object is structured as follows:

```ts
interface AuthRequestPayload {
  token: string;
  tokenType: "JWE";
}
```

Parameter	Type	Required	Default Value	Description
token	*string*	Yes	auth	Authorization token, obtained via HTTP request
tokenType	*string*	Yes	JWE	String *"JWE"*

11.2.5. SuccessAuthResponse

Response from the server upon successful authorization:

```ts
interface SuccessAuthResponse extends ResponseMessage<SuccessAuthResponsePayload> {}
```

The SuccessAuthResponse interface inherits fields from the SuccessAuthResponsePayload object and essentially serves as the message body. Therefore, the field descriptions will be provided there.

11.2.6. SuccessAuthResponsePayload

The message body (payload) for a successful authorization object is as follows:

```ts
interface SuccessAuthResponsePayload {
  userId: string;
}
```

Parameter	Type	Required	Description
userId	*string*	Yes	TrueConf ID of the authorized bot (server user)

11.2.7. ErrorAuthResponse

Response in case of authorization error:

```ts
interface ErrorAuthResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorAuthResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.8. UnathorizedErrorResponse

Response when attempting to send messages without authorization.

```ts
interface UnathorizedErrorResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The UnathorizedErrorResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.9. CreateGroupChatRequest

Interface for creating a group chat:

```ts
interface CreateGroupChatRequest extends RequestMessage<"createGroupChat",CreateGroupChatRequestPayload> {}
```

The CreateGroupChatRequest interface inherits fields from CreateGroupChatRequestPayload, so the field descriptions are provided there.

11.2.10. CreateGroupChatRequestPayload

```ts
interface CreateGroupChatRequestPayload {
  title: string;
}
```

Parameter	Type	Required	Description
title	*string*	Yes	Name of the group chat

11.2.11. SuccessCreateGroupChatResponse

Response upon successful creation of a group chat:

```ts
interface SuccessCreateGroupChatResponse extends ResponseMessage<SuccessCreateGroupChatResponsePayload> {}
```

The SuccessCreateGroupChatResponse interface inherits fields from SuccessCreateGroupChatResponsePayload, so the field descriptions are provided there.

11.2.12. SuccessCreateGroupChatResponsePayload

```ts
interface SuccessCreateGroupChatResponse extends ResponseMessage<SuccessCreateGroupChatResponsePayload> {}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier

11.2.13. ErrorCreateGroupChatResponse

Response in case of an error creating a group chat:

```ts
interface ErrorCreateGroupChatResponse
extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorCreateGroupChatResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.14. RemoveChatRequest

Outgoing command to delete chat:

```ts
interface RemoveChatRequest extends RequestMessage<"removeChat",
RemoveChatRequestPayload> {}
```

The RemoveChatRequest interface inherits fields from RemoveChatRequestPayload, so the field descriptions are provided there.

11.2.15. RemoveChatRequestPayload

The message body (payload) for the remove chat request is as follows:

```ts
interface RemoveChatRequestPayload {
  chatId: string;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier

11.2.16. SuccessRemoveChatResponse

Response upon successful chat deletion:

```ts
interface SuccessRemoveChatResponse
  extends ResponseMessage<SuccessRemoveChatResponsePayload> {}
```

The SuccessRemoveChatResponse interface inherits fields from the SuccessRemoveChatResponsePayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.17. SuccessRemoveChatResponsePayload

```ts
interface SuccessRemoveChatResponsePayload {
  chatId: string;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier

11.2.18. ErrorRemoveChatResponse

```ts
interface ErrorRemoveChatResponse extends GenericErrorResponse<Gener
icErrorPayload<uint32>> {}
```

The ErrorRemoveChatResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.19. AddChatParticipantRequest

Command to add a participant to the chat:

```ts
interface AddChatParticipantRequest extends RequestMessage<"addChatP
articipant",AddChatParticipantRequestPayload> {}
```

The AddChatParticipantRequest interface inherits fields from the AddChatParticipantRequestPayload object. Essentially, it serves as the message body, so the field descriptions will be provided there.

11.2.20. AddChatParticipantRequestPayload

```ts
interface AddChatParticipantRequestPayload {
  chatId: string;
  userId: string;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Identifier of the chat to which the user is being added
userId	*string*	Yes	TrueConf ID of the user being added to the chat

11.2.21. SuccessAddChatParticipantResponse

Response upon successful addition of a participant to the chat:

```ts
interface SuccessAddChatParticipantResponse
  extends ResponseMessage<SuccessAddChatParticipantResponsePayload>
{}
```

The SuccessAddChatParticipantResponse interface inherits fields from SuccessAddChatParticipantResponsePayload, so the field descriptions are provided there.

11.2.22. SuccessAddChatParticipantResponsePayload

The message body (payload) for the successful addition of a participant to the chat object is as follows:

```ts
interface SuccessAddChatParticipantResponsePayload {}
```

11.2.23. ErrorAddChatParticipantResponse

Response in case of an error when removing a participant from the chat:

```ts
interface ErrorAddChatParticipantResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorAddChatParticipantResponse interface inherits fields from GenericErrorResponse, so the description of the fields is provided there.

11.2.24. CreateP2PChatRequest

Command to create a personal chat (peer-to-peer):

```ts
interface CreateP2PChatRequest extends RequestMessage<"createP2PChat
",CreateP2PChatRequestPayload> {}
```

The CreateP2PChatRequest interface inherits fields from the CreateP2PChatRequestPayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.25. CreateP2PChatRequestPayload

```ts
interface CreateP2PChatRequestPayload {
  userId : string;
}
```

Parameter	Type	Required	Description
userId	*string*	Yes	TrueConf ID of the user with whom we want to create a chat

If the bot has never messaged this user before, a new chat will be created. If the bot has previously sent messages to this user, an existing chat will be returned.

11.2.26. SuccessCreateP2PChatResponse

Response upon successful creation of a personal (peer-to-peer) chat:

```ts
interface SuccessCreateP2PChatResponse
  extends ResponseMessage<SuccessCreateP2PChatResponsePayload> {}
```

The SuccessCreateP2PChatResponse interface inherits fields from the CreateP2PChatRequestPayload object. Essentially, it serves as the message body, so the field descriptions will be provided there.

11.2.27. SuccessCreateP2PChatResponsePayload

```ts
interface SuccessCreateP2PChatResponsePayload {
  chatId : string;
}
```

Parameter	Type	Req.	Description
chatId	*string*	Yes	Identifier of the created chat

11.2.28. ErrorCreateP2PChatResponse

```ts
interface ErrorCreateP2PChatResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorCreateP2PChatResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.29. RemoveChatParticipantRequest

Command to remove a participant from the chat:

```ts
interface RemoveChatParticipantRequest
  extends RequestMessage<"removeChatParticipant",RemoveChatParticipa
ntRequestPayload> {}
```

The RemoveChatParticipantRequest interface inherits fields from the RemoveChatParticipantRequestPayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.30. RemoveChatParticipantRequestPayload

```ts
interface RemoveChatParticipantRequestPayload {
  chatId : string;
  userId : string;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Identifier of the chat from which the user is being removed
userId	*string*	Yes	TrueConf ID of the user being removed

11.2.31. SuccessRemoveChatParticipantResponse

Response upon successful removal of a participant from the chat:

```ts
interface SuccessRemoveChatParticipantResponse
  extends ResponseMessage<SuccessRemoveChatParticipantResponsePayloa
d> {}
```

The SuccessRemoveChatParticipantResponse interface inherits fields from the SuccessRemoveChatParticipantResponsePayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.32. SuccessRemoveChatParticipantResponsePayload

```ts
interface SuccessRemoveChatParticipantResponsePayload {}
```

11.2.33. ErrorRemoveChatParticipantResponse

```ts
interface ErrorRemoveChatParticipantResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorRemoveChatParticipantResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.34. HasChatParticipantRequest

Command to check for a participant in the chat:

```ts
interface HasChatParticipantRequest extends RequestMessage<"hasChatP
articipant",HasChatParticipantRequestPayload> {}
```

The HasChatParticipantRequest interface inherits fields from the HasChatParticipantRequestPayload object. Essentially, it serves as the message body, so the field descriptions will be provided there.

11.2.35. HasChatParticipantRequestPayload

```ts
interface HasChatParticipantRequestPayload {
  chatId: string;
  userId: string;
}
```

Parameter	Type	Req.	Description
chatId	*string*	Yes	The chat identifier where the user's presence is checked
userId	*string*	Yes	The TrueConf ID of the user being checked

11.2.36. SuccessHasChatParticipantResponse

Response upon successfully retrieving information about a participant's presence in the chat:

```ts
interface SuccessHasChatParticipantResponse
  extends ResponseMessage<SuccessHasChatParticipantResponsePayload>
{}
```

The SuccessHasChatParticipantResponse interface inherits fields from the SuccessHasChatParticipantResponsePayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.37. SuccessHasChatParticipantResponsePayload

```ts
interface SuccessHasChatParticipantResponsePayload {
  result: boolean;
```

```
}
```

Parameter	Type	Req.	Description
result	*boolean*	Yes	A flag indicating the presence of a participant in the chat. If `true` – the participant is present in the chat

11.2.38. ErrorHasChatParticipantResponse

Response in case of an error when removing a participant from the chat:

```ts
interface ErrorHasChatParticipantResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorHasChatParticipantResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.39. GetChatParticipantListRequest

Command to retrieve the list of chat participants:

```ts
interface GetChatParticipantListRequest extends RequestMessage<"getC
hatParticipants", GetChatParticipantListRequestPayload> {}
```

The GetChatParticipantListRequest interface inherits fields from the GetChatParticipantListRequestPayload object. Essentially, it serves as the message body, so the field descriptions will be provided there.

11.2.40. GetChatParticipantListRequestPayload

```ts
interface GetChatParticipantListRequestPayload {
  chatId: string;
  pageSize: uint32;
  pageNumber: uint32;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier
pageSize	*uint32*	Yes	Number of records with participant information in the response to the request

Parameter	Type	Required	Description
pageNumber	*uint32*	Yes	The offset with which the list of participants will be returned. The offset is calculated using the formula `startOffset = pageSize * pageNumber`

11.2.41. SuccessGetChatParticipantListResponse

Response upon successful retrieval of the participant list in the chat:

```ts
interface SuccessGetChatParticipantListResponse
  extends ResponseMessage<SuccessGetChatParticipantListResponsePaylo
ad> {}
```

The SuccessGetChatParticipantListResponse interface inherits fields from the SuccessGetChatParticipantListResponsePayload object, essentially serving as the message body. Therefore, the field descriptions will be provided there.

11.2.42. SuccessGetChatParticipantListResponsePayload

```ts
interface SuccessGetChatParticipantListResponsePayload {
  participants: Array<ChatParticipant>;
}
```

Parameter	Type	Required	Description
participants	*Array<ChatParticipant>*	Yes	An array of objects containing information about all chat participants

11.2.43. ErrorGetChatParticipantListResponse

Response in case of an error retrieving the list of chat participants:

```ts
interface ErrorGetChatParticipantListResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorGetChatParticipantListResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.44. AddChatParticipantIncomingRequest

Incoming message with information about the new chat participant:


```ts
interface AddChatParticipantIncomingRequest extends RequestMessage<"
addChatParticipant",AddChatParticipantIncomingRequestPayload> {}
```

The AddChatParticipantIncomingRequest interface inherits fields from the AddChatParticipantIncomingRequestPayload object. Essentially, it serves as the message body, so the description of the fields will be provided there.

11.2.45. AddChatParticipantIncomingRequestPayload

```ts
interface AddChatParticipantIncomingRequestPayload {
  chatId: string;
  userId: string;
  addedBy: MessageAuthor;
  timestamp: uint64;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier
userId	*string*	Yes	TrueConf ID of the user added to the chat
addedBy	*MessageAuthor*	Yes	Who added the participant. See the MessageAuthor object for details
timestamp	*uint64*	Yes	Timestamp of when the participant was added to the chat in UNIX timestamp format with millisecond precision

11.2.46. SuccessProceedAddChatParticipantResponse

The response from the client does not contain any payload.

```ts
interface SuccessProceedAddChatParticipantResponse extends ResponseM
essage<{}> {}
```

11.2.47. RemoveChatParticipantIncomingRequest

Incoming message with information about the participant's removal from the chat:

```ts
interface RemoveChatParticipantIncomingRequest
  extends RequestMessage<"removeChatParticipant",
RemoveChatParticipantIncomingRequestPayload> {}
```

The RemoveChatParticipantIncomingRequest interface inherits fields from the RemoveChatParticipantIncomingRequestPayload object. Essentially, it serves as the message body, so the field descriptions will be provided there.

11.2.48. RemoveChatParticipantIncomingRequestPayload

```ts
interface RemoveChatParticipantIncomingRequestPayload {
  chatId: string;
  userId: string;
  removedBy: MessageAuthor;
  timestamp: uint64;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier
userId	*string*	Yes	TrueConf ID of the user removed from the chat
addedBy	*MessageAuthor*	Yes	Who removed the participant. See the MessageAuthor object for details
timestamp	*uint64*	Yes	Timestamp of when the participant was removed from the chat in UNIX timestamp format with millisecond precision

11.2.49. SuccessProceedRemoveChatParticipantResponse

The response from the client does not contain any payload.

```ts
interface SuccessProceedRemoveChatParticipantResponse
  extends ResponseMessage<{}> {}
```

11.2.50. SendChatMessageRequest

Command to send a message:

```ts
interface SendChatMessageRequest extends RequestMessage<"sendMessage
", SendChatMessageRequestPayload> {}
```

The SendChatMessageRequest interface inherits fields from the SendChatMessageRequestPayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.51. SendChatMessageRequestPayload

```ts
interface SendChatMessageRequestPayload {
  chatId: string;
  replyMessageId?: string;
  content: TextMessageContent;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier
replyMessageId	*string*	No	If replying to a message, this field should contain the ID of that message. Optional field
content	*TextMessageContent*	Yes	Message data

11.2.52. TextMessageContent

```ts
interface TextMessageContent {
  text: string;
  parseMode: TextParseModeEnum;
}
```

Parameter	Type	Required	Description
text	*string*	Yes	Message text
parseMode	*TextParseModeEnum*	Yes	Message text parsing mode

11.2.53. SuccessSendChatMessageResponse

Response in case of successful message delivery:

```ts
interface SuccessSendChatMessageResponse extends ResponseMessage<Suc
cessSendChatMessageResponsePayload> {}
```

The SuccessSendChatMessageResponse interface inherits fields from the SuccessSendChatMessageResponsePayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.54. SuccessSendChatMessageResponsePayload

```ts
interface SuccessSendChatMessageResponsePayload {
  chatId: string;
  messageId: string;
  timestamp: uint64;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	The identifier of the chat where the message was sent
messageId	*string*	Yes	The identifier of the message. This identifier can be used later to modify, forward, or delete the message
timestamp	*uint64*	Yes	The timestamp of when the message was sent, in UNIX timestamp format, with millisecond precision

11.2.55. ErrorSendChatMessageResponse

Response in case of a chat message sending error:

```ts
interface ErrorSendChatMessageResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorSendChatMessageResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.56. EditChatMessageRequest

Command to edit the message:

```ts
interface EditChatMessageRequest extends RequestMessage<"editMessage
",EditChatMessageRequestPayload> {}
```

The EditChatMessageRequest interface inherits fields from the EditChatMessageRequestPayload object, as it essentially serves as the message body. Therefore, the field descriptions will be provided there.

11.2.57. EditChatMessageRequestPayload

```ts
interface EditChatMessageRequestPayload {
  messageId: string;
```

```
  content: TextMessageContent;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier
content	*TextMessageContent*	Yes	Message data

11.2.58. SuccessEditChatMessageResponse

Response in case of a successful message update:

```ts
interface SuccessEditChatMessageResponse
  extends ResponseMessage<SuccessEditChatMessageResponsePayload> {}
```

The SuccessEditChatMessageResponse interface inherits fields from the SuccessSendChatMessageResponsePayload object, as it essentially serves as the message body. Therefore, the field descriptions will be provided there.

11.2.59. SuccessEditChatMessageResponsePayload

```ts
interface SuccessSendChatMessageResponsePayload {
  timestamp: uint64;
}
```

Parameter	Type	Req.	Description
timestamp	*uint64*	Yes	Message editing timestamp in UNIX timestamp format with millisecond precision

11.2.60. ErrorEditChatMessageResponse

Response in case of message modification error:

```ts
interface ErrorEditChatMessageResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The interface ErrorEditChatMessageResponse inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.61. ForwardChatMessageRequest

Command to forward a message:

```ts
interface ForwardChatMessageRequest
  extends RequestMessage<"forwardMessage",
ForwardChatMessageRequestPayload> {}
```

The ForwardChatMessageRequest interface inherits fields from the ForwardChatMessageRequestPayload object, essentially serving as the message body. Therefore, the field descriptions will be provided there.

11.2.62. ForwardChatMessageRequestPayload

```ts
interface ForwardChatMessageRequestPayload {
  chatId: string;
  messageId: string;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Identifier of the chat to which you will forward the message
messageId	*string*	Yes	Identifier of the message being forwarded

11.2.63. SuccessForwardChatMessageResponse

Response in case of successful message forwarding:

```ts
interface SuccessForwardChatMessageResponse
  extends ResponseMessage<SuccessForwardChatMessageResponsePayload>
{}
```

The SuccessForwardChatMessageResponse interface inherits fields from the SuccessForwardChatMessageResponsePayload object; essentially, it constitutes the message body, so the field descriptions will be provided there.

11.2.64. SuccessForwardChatMessageResponsePayload

```ts
interface SuccessForwardChatMessageResponsePayload {
  chatId: string;
  messageId: string;
  timestamp: uint64;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	The unique identifier of the chat to which the message was forwarded
messageId	*string*	Yes	The unique identifier of the **new** message containing your forwarded message
timestamp	*uint64*	Yes	The timestamp of the message forwarding in UNIX timestamp format accurate to milliseconds

11.2.65. ErrorForwardChatMessageResponse

Response in case of message forwarding error:

```ts
interface ErrorForwardChatMessageResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorForwardChatMessageResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.66. GetChatMessageRequest

Command to receive a message:

```ts
interface GetChatMessageRequest
  extends RequestMessage<"getMessageById",
GetChatMessageRequestPayload> {}
```

The GetChatMessageRequest interface inherits fields from the GetChatMessageRequestPayload object. Essentially, it serves as the message body, so the description of the fields will be provided there.

11.2.67. GetChatMessageRequestPayload

```ts
interface GetChatMessageRequestPayload {
  messageId: string;
}
```

Parameter	Type	Required	Description
messageId	*string*	Yes	Identifier of the requested message

11.2.68. SuccessGetChatMessageResponse

Response upon successful receipt of the message:

```ts
interface SuccessGetChatMessageResponse
  extends ResponseMessage<SuccessGetChatMessageResponsePayload> {}
```

The SuccessGetChatMessageResponse interface inherits fields from the SuccessGetChatMessageResponsePayload object, as it essentially serves as the message body. Therefore, the field descriptions will be provided there.

11.2.69. SuccessGetChatMessageResponsePayload

```ts
interface SuccessGetChatMessageResponsePayload extends Envelope {}
```

The SuccessGetChatMessageResponsePayload object does not contain its own fields and is inherited from the Envelope object.

11.2.70. ErrorGetChatMessageResponse

Response in case of an error retrieving the message:

```ts
interface ErrorGetChatMessageResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorGetChatMessageResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.71. GetChatHistoryRequest

Command to retrieve chat history:

```ts
interface GetChatHistoryRequest
  extends RequestMessage<"getChatHistory",
GetChatHistoryRequestPayload> {}
```

The GetChatHistoryRequest interface inherits fields from the GetChatHistoryRequestPayload object, essentially serving as the message body, so the description of the fields will be provided there.

11.2.72. GetChatHistoryRequestPayload

```ts
interface GetChatHistoryRequestPayload {
  chatId: string;
  count: uint32;
  fromMessageId?: string;
}
```

Parameter	Type	Req.	Description
chatId	*string*	Yes	The chat ID for which the message history is requested
count	*uint32*	Yes	The number of messages returned in the response
fromMessageId	*string*	No	The chat message ID from which the chat history will be returned

Messages are returned in order from the newest message (or from `fromMessageId`) to the oldest based on the date they were added to the chat.

11.2.73. SuccessGetChatHistoryResponse

```ts
interface SuccessGetChatHistoryResponse
  extends ResponseMessage<SuccessGetChatHistoryResponsePayload> {}
```

The SuccessGetChatHistoryResponse interface inherits fields from the SuccessGetChatHistoryResponsePayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.74. SuccessGetChatHistoryResponsePayload

Response upon successful retrieval of chat history:

```ts
interface SuccessGetChatMessageResponsePayload {
  chatId: string;
  count: uint32;
  messages: Array<Envelope>;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	

Parameter	Type	Required	Description
			The chat identifier for which the message history was requested
count	*uint32*	Yes	The number of received messages
messages	*Array<Envelope>*	No	An array with Envelope objects

11.2.75. ErrorGetChatHistoryResponse

Error response when retrieving chat history:

```ts
interface ErrorGetChatHistoryResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorGetChatHistoryResponse interface inherits fields from GenericErrorResponse, therefore the description of the fields is provided there.

11.2.76. RemoveChatMessageRequest

Command to delete a message:

```ts
interface RemoveChatMessageRequest extends RequestMessage<"removeMes
sage", RemoveChatMessageRequestPayload> {}
```

The interface RemoveChatMessageRequest inherits fields from the RemoveChatMessageRequestPayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.77. RemoveChatMessageRequestPayload

```ts
interface RemoveChatMessageRequestPayload {
  messageId: string;
  forAll: boolean;
}
```

Parameter	Type	Required	Description
messageId	*string*	Yes	Identifier of the message to be deleted
forAll	*boolean*	Yes	Message deletion flag. If set to `true`, the message will be deleted for all users

11.2.78. SuccessRemoveChatMessageResponse

Response upon successful message deletion:

```ts
interface SuccessRemoveChatMessageResponse
  extends ResponseMessage<SuccessRemoveChatMessageResponsePayload>
{}
```

The SuccessRemoveChatMessageResponse interface inherits fields from the SuccessRemoveChatMessageResponsePayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.79. SuccessRemoveChatMessageResponsePayload

```ts
interface SuccessRemoveChatMessageResponsePayload {}
```

11.2.80. ErrorRemoveChatMessageResponse

Response in case of message deletion error:

```ts
interface ErrorRemoveChatMessageResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorRemoveChatMessageResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.81. UploadFileRequest

Command to create a file upload task:

```ts
interface UploadFileRequest
  extends RequestMessage<"uploadFile", UploadFileRequestPayload> {}
```

The UploadFileRequest interface inherits fields from the UploadFileRequestPayload object. Essentially, it serves as the body of the message, so the field descriptions will be provided there.

11.2.82. UploadFileRequestPayload

```ts
interface UploadFileRequestPayload {
  fileName: string;
  fileSize: number;
  mimeType: string;
}
```

Parameter	Type	Required	Description
fileName	*string*	Yes	File name
fileSize	*number*	Yes	File size
mimeType	*string*	Yes	File MIME type

11.2.83. SuccessUploadFileResponse

Response when a file upload task is successfully created:

```ts
interface SuccessUploadFileResponse
  extends ResponseMessage<SuccessUploadFileResponsePayload> {}
```

The SuccessUploadFileResponse interface inherits fields from the SuccessUploadFileResponsePayload object, essentially serving as the message body. Therefore, the field descriptions will be provided there.

11.2.84. SuccessUploadFileResponsePayload

```ts
interface SuccessUploadFileResponsePayload {
  uploadTaskId: string;
}
```

Parameter	Type	Required	Description
uploadTaskId	*string*	Yes	File upload task identifier

11.2.85. ErrorUploadFileResponse

Response in case of an error creating a file upload task:

```ts
interface ErrorUploadFileResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorUploadFileResponse interface inherits fields from GenericErrorResponse, so the description of the fields can be found there.

11.2.86. SendFileRequest

command for sending a file in a message:


```ts
interface SendFileRequest
  extends RequestMessage<"sendFile", SendFileRequestPayload> {}
```

The SendFileRequest interface inherits fields from the SendFileRequestPayload object. Essentially, it is the body of the message, so the field descriptions will be provided there.

11.2.87. SendFileRequestPayload

```ts
interface SendFileRequestPayload {
  chatId: string;
  content: {
    fileId: string,
  };
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier for sending a message
fileId	*string*	Yes	File identifier obtained in the HTTP request response

11.2.88. SuccessSendFileResponse

Response in case of successful file upload:

```ts
interface SuccessSendFileResponse
  extends ResponseMessage<SuccessSendFileResponsePayload> {}
```

The SuccessSendFileResponse interface inherits fields from the SuccessSendFileResponsePayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.89. SuccessSendFileResponsePayload

```ts
interface SuccessSendFileResponsePayload {
  chatId: string;
  timestamp: number;
  messageId: string;
}
```

Parameter	Type	Required	Description
chatId	*string*	Yes	Chat identifier
timestamp	*number*	Yes	Message send time
messageId	*string*	Yes	Message identifier

11.2.90. ErrorSendFileResponse

```ts
interface ErrorSendFileResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorSendFileResponse interface inherits fields from GenericErrorResponse, so the field descriptions can be found there.

11.2.91. FileMessageContent

Receiving file transfer notification:

```ts
interface FileMessageContent {
  name: string;
  mimeType: string;
  size: number;
  fileId: string;
}
```

Parameter	Type	Required	Description
name	*string*	Yes	File name
size	*number*	Yes	File size
mimeType	*string*	Yes	MIME type of the file
fileId	*string*	Yes	File ID on the server

11.2.92. GetFileInfoRequest

Command to obtain file information and download it:

```ts
interface GetFileInfoRequest
  extends RequestMessage<"getFileInfo", GetFileInfoRequestPayload>
{}
```

The GetFileInfoRequest interface inherits fields from the GetFileInfoRequestPayload object, as it essentially serves as the message body. Therefore, the field descriptions will be provided there.

11.2.93. GetFileInfoRequestPayload

```ts
interface GetFileInfoRequestPayload {
  fileId: string;
}
```

Parameter	Type	Required	Description
fileId	*string*	Yes	File identifier obtained from the message sending event

11.2.94. SuccessGetFileInfoResponse

Response when the file is found:

```ts
interface SuccessGetFileInfoResponse
  extends ResponseMessage<SuccessGetFileInfoResponsePayload> {}
```

The SuccessGetFileInfoResponse interface inherits fields from the SuccessGetFileInfoResponsePayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.95. SuccessGetFileInfoResponsePayload

```ts
interface SuccessGetFileInfoResponsePayload {
    name: string,
    size: number,
    mimeType: string,
    downloadUrl: string | null,
    readyState: FileReadyStateEnum,
    infoHash: string,
    previews: null | Array<{
      name: string,
      mimeType: string,
      size: number,
      downloadUrl: string,
    }>,
  }
```

Parameter	Type	Req.	Description
name	*string*	Yes	File name
size	*number*	Yes	File size in bytes
mimeType	*string*	Yes	File MIME type
downloadUrl	*string*	Yes	URL for file download
readyState	*FileReadyStateEnum*	Yes	File status on server
infoHash	*string*	Yes	File identifier on server
previews	*Array*	Yes	Array of file previews

* To download the file, you need to send a GET request to the address specified in `downloadUrl`.

11.2.96. ErrorGetFileInfoResponse

Response if the file is not found on the server:

```ts
interface ErrorGetFileInfoResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorGetFileInfoResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.97. SubscribeFileProgressRequest

Command to subscribe to file upload progress on the server:

```ts
interface SubscribeFileProgressRequest extends RequestMessage<"subsc
ribeFileProgress", SubscribeFileProgressRequestPayload> {}
```

The SubscribeFileProgressRequest interface inherits fields from the SubscribeFileProgressRequestPayload object, essentially forming the message body. Therefore, the field descriptions will be provided there.

11.2.98. SubscribeFileProgressRequestPayload

```ts
interface SubscribeFileProgressRequestPayload {
  fileId: string;
}
```

Parameter	Type	Required	Description
fileId	*string*	Yes	File identifier on the server

11.2.99. SuccessSubscribeFileProgressResponse

Response in case of a successful subscription:

```ts
interface SuccessSubscribeFileProgressResponse
  extends ResponseMessage<SuccessSubscribeFileProgressResponsePayloa
d> {}
```

The SuccessSubscribeFileProgressResponse interface inherits fields from the SuccessSubscribeFileProgressResponsePayload object, essentially serving as the message body. Therefore, the field descriptions will be provided there.

11.2.100. SuccessSubscribeFileProgressResponsePayload

```ts
interface SuccessSubscribeFileProgressResponsePayload {
  result: boolean;
}
```

Parameter	Type	Req.	Description
result	*boolean*	Yes	Whether the subscription to download was successful

11.2.101. ErrorSubscribeFileProgressResponse

Response in case of a failed file upload subscription:

```ts
interface ErrorSubscribeFileProgressResponse extends GenericErrorRes
ponse<GenericErrorPayload<uint32>> {}
```

The ErrorSubscribeFileProgressResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.102. UploadProgressContent

File upload event message interface:

```ts
interface UploadProgressContent {
  fileId: string;
  progress: number;
}
```

Parameter	Type	Required	Description
fileId	*string*	Yes	File identifier on the server
progress	*number*	Yes	Number of bytes uploaded to the server

11.2.103. UnsubscribeFileProgressRequest

Command to unsubscribe from receiving file upload events to the server.

```ts
interface UnsubscribeFileProgressRequest extends RequestMessage<"uns
ubscribeFileProgress", UnsubscribeFileProgressRequestPayload> {}
```

The UnsubscribeFileProgressRequest interface inherits fields from the UnsubscribeFileProgressRequestPayload object, essentially serving as the message body, so the field descriptions will be provided there.

11.2.104. UnsubscribeFileProgressRequestPayload

```ts
interface UnsubscribeFileProgressRequestPayload {
  fileId: string;
}
```

Parameter	Type	Required	Description
id	*string*	Yes	File identifier on the server

11.2.105. SuccessUnsubscribeFileProgressResponse

Response upon successful unsubscription from receiving file upload events to the server:

```ts
interface SuccessUnsubscribeFileProgressResponse extends ResponseMes
sage<SuccessUnsubscribeFileProgressResponsePayload> {}
```

The SuccessUnsubscribeFileProgressResponse interface inherits fields from the SuccessUnsubscribeFileProgressResponsePayload object, essentially serving as the body of the message. Therefore, the field descriptions will be provided there.

11.2.106. SuccessUnsubscribeFileProgressResponsePayload

```ts
interface SuccessUnsubscribeFileProgressResponsePayload {
  result: boolean;
}
```

Parameter	Type	Required	Description
result	*boolean*	Yes	Indicates if unsubscribing from receiving events was successful

11.2.107. ErrorUnsubscribeFileProgressResponse

Response in case of an error when unsubscribing from receiving file upload events on the server:

```ts
interface ErrorUnsubscribeFileProgressResponse
  extends GenericErrorResponse<GenericErrorPayload<uint32>> {}
```

The ErrorUnubscribeFileProgressResponse interface inherits fields from GenericErrorResponse, so the field descriptions are provided there.

11.2.108. Envelope

Each message received as an event, loaded by message ID, or retrieved from the chat message history conforms to the Envelope interface:

```ts
interface Envelope {
  messageId: string;
  chatId: string;
  timestamp: number;
  replyMessageId?: string;
  isEdited: boolean;
  type: EnvelopeTypeEnum;
  author: EnvelopeAuthor;
  box: EnvelopeBox;
  content: AbstractEnvelopeContent;
}
```

Parameter	Type	Required	Description
messageId	*string*	Yes	Unique identifier of the message
chatId	string	Yes	Unique identifier of the chat containing the message
timestamp	uint64	Yes	Message sending timestamp in UNIX timestamp format with millisecond precision
replyMessageId	string	No	If it's a reply to a message, contains the identifier of that message
isEdited	boolean	Yes	Whether the message has ever been edited
type	EnvelopeTypeEnum	Yes	Type of message
author	EnvelopeAuthor	Yes	Message author
box	EnvelopeBox	Yes	Sequential numbers of the message
content	AbstractEnvelopeContent	Yes	Message content. It can be text, a file, or another type of data, depending on the `type` field.

11.2.109. EnvelopeAuthor

The object contains information about the message author.

```ts
interface EnvelopeAuthor {
  id: string;
  type: EnvelopeAuthorTypeEnum;
}
```

Parameter	Type	Required	Description
id	string	Yes	A unique identifier for the message author. This can be either the TrueConf ID of the user who sent the message or the server name from which it was sent. The type of identifier depends on the value of the `type` field.
type	EnvelopeAuthorTypeEnum	Yes	Type of message author

11.2.110. EnvelopeBox

This object is used for sorting messages and has the following interface:

```ts
interface EnvelopeBox {
  id: number;
  position: string;
}
```

Parameter	Type	Req.	Description
id	number	Yes	Sequential number of the "box" containing the message
position	string	Yes	Sequential number of the message in the "box"

You can learn how to work with this object in the corresponding section.

11.2.111. GenericErrorResponse

For most requests, in case of an error, a response will be sent in the following format:

```ts
interface GenericErrorResponse<T extends GenericErrorPayload> extend
s ResponseMessage<T> { }
```

The GenericErrorResponse interface inherits fields from GenericErrorPayload and ResponseMessage, so refer to their field descriptions there.

11.2.112. GenericErrorPayload

```ts
interface GenericErrorPayload<ErrorCodeEnum extends uint32> {
    errorCode: ErrorCodeEnum ;
}
```

Parameter	Type	Required	Description
errorCode	uint32	Yes	Error code

11.2.113. Message

Basic message format. Any message (both incoming and outgoing) will have the following fields:

```ts
interface Message {
  type: MESSAGE_TYPE;
  id: uint64;
}
```

Parameter	Type	Required	Description
type	enum MESSAGE_TYPE	Yes	Message type from the `MESSAGE_TYPE` enumeration
id	uint32	Yes	Unique message identifier, represented as an incrementing number (in some protocols, such as TCP, this number is called a **sequence id**). The identifier can be identical for outgoing and incoming messages. The identifier value may increment by more than one at a time. See details below.

11.2.114. RequestMessage

A message of type Request.

```ts
interface RequestMessage<Method extends string, Payload extends
JSONObject> extends Message {
  type: MESSAGE_TYPE.REQUEST;
  method: Method;
  payload: Payload;
}
```

Parameter	Type	Required	Description
type	MESSAGE_TYPE.REQUEST	Yes	Message type (default is `1`).
method	string	Yes	Type of command being sent
payload	JSONObject	No	Any JSON-formatted object applicable for this ***method***. This field may be omitted if no additional parameters are specified for the ***method***.

Example:

```json
{
  "type": 1,
```

```
  "id": 1,
  "method": "ping",
  "payload": {
    "data": "Hello, World!",
    "some_additional_data": {
      "internal_additional_field_124": [true, false, 1234]
    }
  }
}
```

11.2.115. ResponseMessage

Message with the type Response.

```ts
interface ResponseMessage<Payload extends JSONObject> extends Messag
e {
  type: MESSAGE_TYPE.RESPONSE;
  payload: Payload;
}
```

Parameter	Type	Req.	Description
type	MESSAGE_TYPE.RESPONSE	Yes	Message type (default is `2`).
payload	JSONObject	No	Any JSON-formatted object applicable as payload data for the request response. This field may be omitted if the response does not contain any payload data.

Example:

```json
{
  "type": 2,
  "id": 123456,
  "payload": {
    "data": "Hello, World!"
  }
}
```

11.2.116. JSONObject

The JSONObject object passed in the payload should match the following type:

```ts
type JSONValue = null | string | number | boolean | JSONArray |
JSONObject;
interface JSONArray extends Array<JSONValue> {}

type JSONObject = {
    [key: string] ?: JSONValue;
    [key: number] ?: JSONValue;
}
```

11.2.117. OAuthTokenRequest

Request structure for obtaining an identification token.

```ts
interface OAuthTokenRequest {
  client_id: string;
  grant_type: string;
  username: string;
  password: string;
}
```

Parameter	Type	Required	Default	Description
client_id	*string*	Yes	"chat_bot"	Constant string "chat_bot"
grant_type	*string*	Yes	"password"	Constant string "password"
username	*string*	Yes		User login (see below)
password	*string*	Yes		User password

TrueConf server user login. The following formats are supported:

- **Simple login** — the username of the current server user, for example: `vasya` or `alex_smith`
- **With the TrueConf server specified** — for example: `vasya@server1.local`
- **With LDAP domain specification** — for example: `web-department.mycompany\\alex_smith`
- **With server and LDAP domain specified** — for example: `web-department.mycompany\\alex_smith@server1.local`

11.2.118. OAuthTokenSuccessResponse

Response structure upon successful authorization when receiving an access token.

```ts
interface OAuthTokenSuccessResponse {
  access_token: string;
  token_type: string;
  expires_in: uint32;
}
```

Parameter	Type	Required	Default	Description
access_token	*string*	Yes		The identification token used for WebSocket connection authorization.
token_type	*string*	Yes	"JWE"	Constant string "JWE"
expires_in	*uint32*	Yes		The token's lifespan in seconds from the moment it is issued is **one year**. After this period, re-authorization is required.

11.2.119. OAuthTokenErrorResponse

Response structure in case of an authorization error when retrieving the access token.

```ts
interface OAuthTokenErrorResponse {
  error: "OAUTH_ERROR";
  error_description: string;
}
```

Parameter	Type	Required	Description
error	*enum OAUTH_ERROR*	Yes	A value from the OAUTH_ERROR enumeration
error_description	*string*	Yes	Human-readable text of the authorization error

11.2.120. RemoveParticipantEnvelopeContent

This object is used as the content of a message with the type EnvelopeTypeEnum.REMOVE_PARTICIPANT, which is sent when a participant is removed from the chat, and has the following interface:

```ts
interface RemoveParticipantEnvelopeContent extends AbstractEnvelopeC
ontent {
  userId: string;
```

```
}
```

Parameter	Type	Required	Description
userId	*string*	Yes	TrueConf ID of the user removed from the chat.

11.2.121. PlainMessageEnvelopeContent

This object is used as the content of a message with the type EnvelopeTypeEnum.PLAIN_MESSAGE and has the following interface:

```ts
interface RemoveParticipantEnvelopeContent extends AbstractEnvelopeC
ontent {
  text: string;
  parseMode: TextParseModeEnum;
}
```

Parameter	Type	Required	Description
text	*string*	Yes	Message text
parseMode	*TextParseModeEnum*	Yes	Message text parsing mode. See the enumeration description for details

Example:

```json
{
  "chatId": "51ffe1b1-1515-498e-8501-233116adf9da",
  "messageId": "fdfa8fd6-b4f5-4ecc-8c7e-11e420105056",
  "timestamp": 1735330910,
  "author": {
    "id": "user@video.example.com",
    "type": 1 //0 - server, 1 - user
  },
  "replyMessageId": null, //string if this message is a reply
  "isEdited": false, //true if edited
  "box": {
    "id": 1,
    "position": "" //for sorting undelivered messages
  },
  "type": 200, //200 - PLAIN_MESSAGE
  "content": {
```

```
    "text": "Hello, chat bot!",
    "parseMode": "text"
  }
}
```

11.2.122. ForwardMessageEnvelopeContent

This object is used as message content with the type EnvelopeTypeEnum.FORWARD_MESSAGE and has the following interface:

```ts
interface ForwardMessageEnvelopeContent extends Omit<Envelope,
"box">, AbstractEnvelopeContent {}
```

A distinctive feature of the interface for the object with the content of a forwarded message is that it contains the forwarded message object (Envelope) as its content, except for the data in the "box" field.

Theoretically, the chain of nested forwarded messages can be infinite, but currently, the nesting is limited to one message (Envelope).

Example:

```json
{
  "chatId": "51ffe1b1-1515-498e-8501-233116adf9da",
  "messageId": "552a5a7c-02fd-4c30-a4f1-60c9cfff649c",
  "timestamp": 1735330914,
  "author": {
    "id": "user@video.example.com",
    "type": 1 //0 - server, 1 - user
  },
  "replyMessageId": null, //string if this message is reply
  "isEdited": false, //true if edited
  "box": {
    "id": 5677289,
    "position": "COFFEE" //for sorting undelivered messages
  },
  "type": 201, //201 - FORWARD_MESSAGE
  "content": {
    "chatId": "51ffe1b1-1515-498e-8501-233116adf9da",
    "messageId": "fdfa8fd6-b4f5-4ecc-8c7e-11e420105056",
    "timestamp": 1735330910,
    "author": {
```

```
      "id": "user@video.example.com",
      "type": 1 //0 - server, 1 - user
    },
    "replyMessageId": null, //string if this message is reply
    "isEdited": false, //true if edited
    "type": 200, //200 - PLAIN_MESSAGE
    "content": {
      "text": "Hello, chat bot!",
      "parseMode": "text"
    }
  }
}
```

11.2.123. AttachmentMessageEnvelopeContent

This object is used as the content of a message with the type EnvelopeTypeEnum.ATTACHMENT and has the following interface:

```ts
interface AttachmentMessageEnvelopeContent extends AbstractEnvelopeC
ontent {
  name: string;
  mimeType: string;
  size: number;
  id: string;
}
```

Parameter	Type	Required	Default	Description
name	*string*	Yes		The name of the file on the sender's side. For example, "photo.jpg"
mimeType	*string*	Yes	""	The MIME type of the file being sent. It could be a string such as "image/jpeg" or an empty string if the attachment type could not be determined
size	*uint64*	Yes		The file size in bytes
id	*string*	Yes	""	A unique identifier for the file to enable downloading. For more details, see the section on working with attachments

11.2.124. ChatParticipant

This object is used to store information about a chat participant.

```ts
interface ChatParticipant {
  userId: string;
  role: ChatParticipantRoleEnum;
}
```

Parameter	Type	Req.	Description
userId	*string*	Yes	User's TrueConf ID added to the chat
role	*ChatParticipantRoleEnum*	Yes	User's role in the chat (see ChatParticipantRoleEnum enumeration)