API manual

This API is designed to work with Chat2Desk service.

LIST OF CHANGES

Date	Version	Changes
11.02.2021	1.54 for v2.48.0	Added parameter *domain_urls* to companies/api_info (GET) Added parameter *post_url* in *extra_data* to messages (GET). Added error response specification
22.12.2020	1.53	Added new webhook events: – *delete_tag_from_request* – *client_updated* Added new parameter *avatar* to operators (GET)
08.10.2020	1.52	Added new *dialog_transferred* event for which webhooks will be triggered: the dialog is assigned to an operator or assigned operator is changed.
01.10.2020	1.51	Changed messages (POST) *operator_id* info Added new parameter *all_client_channels* in *add_tag_to_client* webhook settings.
17.06.2020	1.50	Info about checking webhook changed.
21.04.2020	1.49	Added information about current *status_id* to operators (GET). Added new parameter *start_id* to messages (GET).
25.03.2020	1.48	Added parameter *client_id* to web_analytics_data (GET).
10.03.2020	1.47	Added *external_id* to messages (POST).
25.02.2020	1.46	Added information about current webhook status and reason of deactivation to webhooks (GET). Added parameter to activate/deactivate webhooks to webhooks (PUT).
25.12.2019	1.45	Added client's roistat id to method web_analytics_data (GET)
21.11.2019	1.44	Added new command companies/switch_mode (PUT) for changing company's current work-mode.
03.10.2019	1.43	Field *offline_type* fixed in operators (GET)
01.10.2019	1.42	Now we resend web hook 3 times in case it fails. statistics (GET) now supports *limit* and *offset* parameters.
6.09.2019	1.41	Added new command operators (PUT) for changing operator's working status. Added new command to get all available operators' working statuses available in the system — operators/statuses (GET). Added new command web_analytics_data (GET) to receive URL and UTM track as well as Google, Yandex and Comagic user ids of your clients. Added encryption option to messages (POST). Added *comment* message type to messages (POST) for sending internal text comments to chat. Added 3 new filters to messages (GET).
23.07.2019	1.40	All webhook requests now contain *event_time*.

		External id is now supported in messages/inbox (POST) and webhook (*inbox*). *Extra data* with *external id* info added to messages (GET). requests (GET) can now return request info (tags and dialog id). Added ability to assign and delete tags to/from *request* using tags/assign_to (POST) and tags (DELETE).
17.06.2019	1.39	Added new parameter *order* to change the order (ascending or descending) of records returned in messages (GET), clients (GET) and dialogs (GET).
15.06.2019	1.38	Added new method to close a request – requests/close (PUT).
4.06.2019	1.37	Added *client_id* parameter in dialogs (GET).
22.05.2019	1.36	Added option for working with **AI** providers in prompter mode: new web hook event *ai_hints_requested*. See webhooks_POST. Added new method for getting **aggregated statistics** data – statistics (GET). Added option to create a new a client via email ("Write first") using clients (POST).
29.04.2019	1.35	Added color buttons support for our online chat using messages (POST).
22.03.2019	1.34	Added *insta_comment* (0 or 1) parameter to messages (GET) and webhook which distinguishes Instagram comments from Direct messages.
7.03.2019	1.33	Added *Write first* feature status and API calls limit info to api_info (GET). Added operator *external_id* to operators (GET).
26.02.2019	1.32	Added *channel_id* filter to channels (GET).
22.02.2019	1.31	Added *close_request* event to webhooks (POST).
6.02.2019	1.30	Added *client_id* to *close_dialog* webhook event. Operator is now considered online only if he or she has first («Working») status – see operators (GET).
14.12.2018	1.29	Added *add_tag_to_request* event to webhooks (POST).
3.12.2018	1.28	Added scheme illustration of external channel.
13.11.2018	1.27	Added *avatar* parameter to .
26.10.2018	1.26	Added examples of webhook data. Also added dedicated URL to test web hooks.
16.10.2018	1.25	Added new methods for working with self-service menu: • menu_items (GET) • send_menu_item (POST)
23.09.2018	1.24	Added new Postman link to open test API commands on the web. All commands are now grouped in Postman.
12.09.2018	1.23	Added *attachments* parameter description in messages (GET) Added *client_phone* parameter description in send_menu_item (POST)
28.08.2018	1.22	• Parameter *nickname* added for clients (POST) • New method added – companies/api_info (GET)
11.08.2018	1.21	New methods added: • webhooks (POST) • webhooks (GET) • webhooks/id (PUT) • webhooks/id (DELETE) • requests/id/messages (GET) • custom_client_fields (GET)

TERMS

A client – is a person, who contacts your company via Chat2Desk service using messengers, online chat, external channel or SMS.

A channel – is an account (phone number or id) in messengers WhatsApp, Viber, Telegram, Facebook, Instagram, VKontakte, Online Chat, SMS and others or external channel, used by your company to chat with clients via service Chat2Desk. You may have many channels with different messengers' accounts on each channel.

An external channel – is a source of messages other than channel specified above, which is not directly connected to the Chat2Desk service. For example, 3rd party live chat, your own messenger or your own CRM. Such external channel can be connected to the service using this API.

A transport – is a messenger, social network or SMS, using which, messages are being received or sent via given channel.

GENERAL INFORMATION

1. There are many ways to work with Chat2Desk. All of them are described in Developers starting guide.
2. Use this link to access a collection of preset API commands for Chat2Desk in **Postman app**:

   https://documenter.getpostman.com/view/9553101/SW7gSQAM

   You must input your *API token* to use these commands (see item 6 below).

3. RESTful architecture and methods GET, PUT, DELETE and POST are used.
4. Commands' results are presented in JSON format.
5. This API uses HTTPS protocol.
6. Authorization is based on *token*, which is available in *Settings/API* on Chat2Desk site (under admin). Here are the parameters for all commands:

```
Authorization: <token>
Host: https://api.chat2desk.com or https://api.chat24.io
```

7. Commands that return large list of data support pagination using URL parameters:
   a. *limit* – number of records returned (maximum is 200, default is 20).
   b. *offset* – offset from the 1st record of the list (default is 0).

   Example of the request (list of clients): */v1/clients?offset=50&limit=30*

8. In general, the system is designed to answer those clients who first contacted your company and so are in the list of you clients. But using clients (POST) you can create a new client and then contact him or her without their first message.

9. If error occurs when request is executed, detailed response with type and description of error will be given.

   Parameters:

   - *message* – type of error;
     - 400 - bad_request;
     - 401 – unauthorized;
     - 403 – forbidden;
     - 404- not found;
     - 500 - unprocessable error.
   - *errors* – detailed description.

   Examples:

```
{
    "message": "bad_request",
    "errors": {
        "tag_group_id": [
            "Tags group with id #1111 not found or does not
             belong to your company"
        ]
    },
    "status": "error"
```

```
{
    "message": "not_found",
    "errors": "Dialog #1111 not found",
    "status": "error"
}
```

BEFORE YOU BEGIN

1. Obtain your <*token*> on Chat2Desk site in *Settings/API*. Also make sure that your API-access level is at least *demo*. This information is also available with api_modes (GET).

2. There are 2 ways to work with messages:

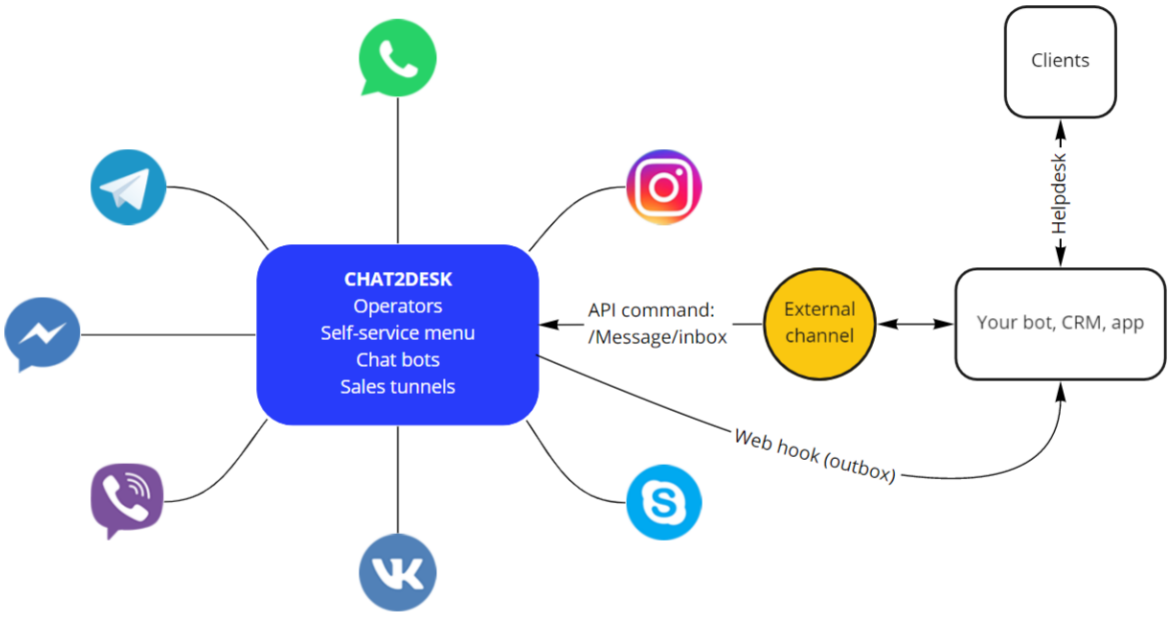
   - Your clients use regular messengers to send messages to your company. In this case use webhooks (POST) or messages (GET) to **receive** messages from the clients.

     To **send** messages use:

     - clients (POST) to create a new client that never texted you before ("Write first" option). This method is needed to send the client a message right after, because the system requires a *client id* to send a message.

     - messages (POST) to send message to existing client that either texted you by his/her initiative or was created using method above.

   - You use *external channel* to pass messages to Chat2Desk from some external source of messages (like CRM). In this case you should operate messages/inbox (POST) to transfer messages from your clients to the system. To receive operators' replies and send them back to your clients – you should use web hooks (see webhooks (POST)). See illustration below.



LIST OF API COMMANDS

1	api_modes (GET)	Returns a list of available API access levels.
2	channels (GET)	Returns a list of channels with info.
3	channels/<id>/clients (GET)	Returns a list of clients by specified channel.
4	clients (GET)	Returns a list of clients with their info.
5	clients (POST)	Creates a new client so you can contact this client afterwards.
6	clients (PUT)	Assigns a name to a client.
7	companies/api_info (GET)	Returns current API version, your API access level, number of API requests per month and more.
8	companies (PUT)	Stores company's custom data.
9	companies/switch_mode (PUT)	Switches company's current work-mode
10	countries (GET)	Returns a list of available countries.
11	custom_client_fields (GET)	Returns a list of custom fields of client info card.
12	delete_outbox (GET)	Deletes all outgoing messages that yet have not been sent
13	dialog_states (GET)	Returns a list of available dialog states.
14	dialogs (GET)	Returns a list of dialogs.
15	dialogs (PUT)	Changes a dialog state and sets up its operator.
16	message_types (GET)	Returns a list of available message types
17	messages (GET)	Returns a list of accumulated messages (both from and to clients).
18	messages (POST)	Sends a message to a client.

19	messages/<id>/transfer (GET)	Assigns a message and corresponding dialog to an operator.
20	messages/inbox (POST)	Sends incoming message from client to the system using *external channel*.
21	messages/read (GET)	Tags a message as read or unread by operator.
22	operators (GET)	Returns a list of operators in the system.
23	operators (PUT)	Changes operator's working status.
24	operators_groups (GET)	Returns a list of operator's groups in the system.
25	qr-decode (POST)	Decodes QR-code.
26	questions/<id>/send (GET)	Sends specified menu item to the client when using scripts (see *Help > Scripts manual*).
27	regions (GET)	Returns a list of available regions.
28	requests (GET)	Returns a list of messages for specified requests.
29	requests/close (PUT)	Closes a request.
30	roles (GET)	Returns a list of available roles.
31	scenarios (GET)	Returns a list of available self-service menu items.
32	scenarios (POST)	Sends self-service menu item to a client.
33	statistics (GET)	Returns aggregated statistics.
34	tag_groups (POST)	Creates a new tag group.
35	tags (DELETE)	Deletes client tag.
36	tags (GET)	Returns a list of tags.
37	tags (POST)	Creates a new tag.
38	tags/assign_to (POST)	Assigns one or more tags to a client.
39	templates (GET)	Returns a list of templates.

40	transfer_to_group (GET)	Transfers (arranges) a dialog on group of operators.
41	transports (GET)	Returns a list of available transports and their supported attachments.
42	web_analytics_data (GET) New	Returns a list of URLs and UTM marks as well as Google, Yandex and Comagic user ids of your clients.
43	webhooks (DELETE)	Deletes specified web hook.
44	webhooks (GET)	Returns a list of web hooks or info about specified web hook.
45	webhooks (POST)	Creates a new web hook for various events.
46	web_hook (POST) Depriciated	Old method to set up web hook. Use webhooks (POST) instead.
47	webhooks (PUT)	Updates specified web hook.

api_modes (GET)

Returns a list of available API access levels. To check your current access level – see api_info (GET).

Request:

```
GET /v1/help/api_modes
```

Params:

- <token> – here and everywhere see description of authorization.

Typical reply:

```
{
"data": [
    {
      "name": "disabled",
      "description": "API disabled"
    },
    {
      "name": "demo",
      "description": "1000 API requests per month"
    },
    {
      "name": "normal",
      "description": "250000 API requests per month"
    },
    {
      "name": "unlimited",
      "description": "unlimited API requests per month"
    }
  ],"status": "success"
}
```

channels (GET)

Returns list of channels with info.

Request:

```
GET /v1/channels/<id>
```

This filter is supported:

- phone
- channel id

Typical request with the filter:

```
GET /v1/channels?phone=172502619555

GET /v1/channels/123
```

Typical reply:


```
{
    "data": [
        {
            "id": 21288,
            "name": "Sales",
            "phone": "1112233",
            "transports": [
                "telegram",
                "facebook",
                "widget",
                "vk",
                "insta_i2crm",
                "ok"
            ]
        }
    ],
    "meta": {
        "total": 1,
        "limit": 20,
        "offset": 0
    },
    "status": "success"
}
```


clients (GET)

Returns a list of clients with their info.

Request:

```
GET /v1/clients
or
GET /v1/clients/<id>
or
GET /v1/clients/<id>/transport
or
GET /v1/clients?phone=79310001122
or
GET /v1/clients/<id>/last_question
or
GET /v1/clients/<id>/dialogs
or
GET /v1/clients/?tags=1,2,5
or
GET /v1/clients/?created_after=1522950064
Or
GET /v1/clients/<id>/questions?start_date=
10-09-2011&finish_date=10-09-2019
```

Params:

- <id> – client's id. If omitted, full list of clients is returned.
- If *transport* key is specified, then the list of available transports for the client specified is returned.
- If *last_question* key is specified, then the last menu item sent to the client is returned.
- If *dialogs* key is specified, then the list of the client's dialogs is returned.
- If questions key is specified, then a list of menu items that were sent to a client when using scripts is returned. Optionally you can specify date span.
- phone – client's phone filter.
- tags – tags filter, comma delimited. Clients with at least 1 tag from the list are returned.
- created_after – clients with creation date (by the first message) after specified unix time are returned.
- order (*asc* or *desc*) – order of records returned. Default is ascending order.

If *id* or *phone* are omitted, full list of clients is returned.

Typical reply without specifying client id (full list of clients):


```
{"data": [
…
    {
      "id": 1607,
      "comment": "This is a comment",
      "assigned_name": "Real VIP client",
      "phone": "19001112233",
      "name": "Thats my name!",
      "avatar":
"http://chat2desk.com/images/users/client/1112233_1.jpg"
,
      "region_id": null,
      "country_id": 1
    },
…
  ],
  "meta": {
    "total": 3911,
    "limit": 20,
    "offset": 1600
  },
  "status": "success"}
```


Typical reply when client id is specified (one client with extended info):


```
{"data": {
      "id": 1607,
      "comment": "This is a comment",
      "assigned_name": "Real VIP client",
      "phone": "19001112233",
      "name": "Thats my name!",
      "avatar":
"http://chat2desk.com/images/users/client/11112233_1.jpg",
      "region_id": null,
      "country_id": 1
      "first_client_message": "2015-12-27T18:36:41 UTC",
      "last_client_message": "2016-01-28T20:29:25 UTC"
  },
  "status": "success"}
```


Some fields:

- assigned_name – a name assigned to the client – see clients (PUT).
- comment – commentary to client left by an operator on the site.
- custom_fields – json data of your client's custom fields with their id, name and value. These fields are visible to operators in client's info card. They are managed by admin in *Settings* > *General* section.
- phone – client's phone number if exists. Available only for WhatsApp, Viber, Viber Business and SMS. For other messengers, these messenger's respective client id is stored. Also see *client_phone* below.

- client_phone – client's phone if the client revealed his/her real phone number in messengers that do not disclose it by default (currently Telegram and Facebook).
- name – client's nickname (if available).

See below a typical reply when specifying client id and transport key. Actually, these are the transports (with channels), which given client used to chat with your chat center and which can be used to answer to that client.


```
{"data": [
    {
      "channel_id": 1,
      "transports": [
        "sms",
        "whatsapp"
      ]
    },
    {
      "channel_id": 2,
      "transports": [
        "viber"
      ]
    }
  ],
  "status": "success"}
```


See below a typical reply when specifying client id and last_question key. This request returns last menu item that was sent to this client earlier.


```
{"data": {
    "id": 4322,
    "text": "This is menu item #20",
    "image": null
  },
  "status": "success"}
```


clients (POST)

Creates a new client to send a message to him or her. This might be useful, for example, to send WhatsApp notifications to your clients who never contacted you before.

Use with caution because a client that you contact first, likely, doesn't have your number in address book and can tap "This is spam!" button, which may lead your WhatsApp or Viber account to get blocked.

Request:

```
POST
/v1/clients?phone=19310001122&transport=whatsapp&channel_id=1
&nickname=Peter
```

Params:

- phone – client's phone number with country code, without any symbols and spaces or email address if you want to contact the client via email.
- transport – name of a transport (messenger), via which you want to contact this client. You can create a new client only using WhatsApp, Viber, Viber Business, Email and SMS. To get a list of available transports use transports (GET).
- channel_id – (optional) id of a channel in which the client will be created. If left blank, your first channel will be used. To get a list of your channels use channels (GET).
- nickname – (optional) assigned name of this new client.

After successful execution, the command returns an info on newly created client as shown below. Region and country are determined based on the phone number.

```
{
"data": {
    "id": 75,
    "assigned_name": null,
    "phone": "11111111113",
    "name": null,
    "avatar": null,
    "region_id": null,
    "country_id": 1,
    "first_client_message": "2016-07-
07T05:34:28 UTC",
    "last_client_message": "2016-07-
07T05:34:28 UTC"
  },
  "status": "success"
}
```

Having received the client's id, you can now send this client a message using messages (POST).

The option to create a new client to message him or her afterwards ("Write first" option) is turned off by default. In this case the command will return an error. Contact administration to enable this option.

clients (PUT)

Assigns a name (assigned name), comment and sets extra fields to a client.

Request:

```
PUT /v1/clients/<id>
```

Request body:

```
{"nickname": "Peter",
 "comment": "This is comment",
 "custom_fields": {"1":"Field 1","2":"Field 2","3":"Field
3"}}
```

Params:

- <id> – client id.
- nickname – client's name (assigned name). If *null* is specified, then the name will be deleted.
- custom fields – json of custom fields ids and their values.
- external_id – client id to use in external system.
- external_service – external service for external client id. Now allowed only "amo" and "bitrix".

Typical reply:

```
{
    "status": "success"
}
```

companies/api_info (GET)

Returns:

- Current API version used by our service.
- Your current API access level – see api_modes (GET).
- Number of available channels
- Custom info (see companies (PUT))
- *Write first* feature status

- URLs of domains to open user interface
- API calls limit and other parameters.

Request:

```
PUT /v1/ companies/api_info
```

Typical reply:


```
{
    "data": {
        "mode": "demo",
        "company_mode": "enterprise",
        "hook": null,
        "hook_preferences": null,
        "current_version": 1,
        "requests_this_month": 72,
        "channels": 5,
        "company_name": "Test",
        "admin_email": "example@example.com",
        "custom_info": {
            "1": "Field 1"
        },
        "write_first_option": false,
        "domain_urls": ["http://web.chat2desk.com",
                        "http://vip.chat2desk.com"]
        "api_calls": {
            "limit_per_month": 1000,
            "used_this_month": 72
        }
    },
    "status": "success"
}
```


companies (PUT)

This command is used to store some text info which relates to your company as a whole – unlike per-client basis.

Request:

```
PUT /v1/companies
```

Request body:

```
{"custom_fields": {"1":"Field 1","2":"Field 2","3":"Field
3"}}
```

Custom data is stored in json-format. It can be read back using companies/api_info (GET).

Typical reply:

```
{
    "status": "success"
}
```

companies/switch_mode (PUT) New

This command is used to switch company's current work-mode to "Working" or "Offline".

Request:

```
PUT /v1/companies/switch_mode
```

Request body:

```
{
    "mode": "online"
}
```

Valid modes: *online* or *offline*.

countries (GET)

Returns a list of available countries in the system.

Request:

```
GET /v1/countries
```

Typical reply:

```
{
    "data": [
        {
            "id": 1,
            "name": "USA, Canada"
        },
        {
            "id": 2,
            "name": "Russia"
        }
    ],
    "meta": {
```

```
            "total": 218,
            "limit": 2,
            "offset": 0
        },
        "status": "success"
    }
```


custom_client_fields (GET)

Returns a list of custom fields of client info card.

Request:

```
GET /v1/custom_client_fields
```

Some fields in the results:

- type – *input* (free text) or *dropdown* (preset values).
- value – preset values for *dropdown* type.
- editable – if *true*, operators can edit this field manually in client's info card.
- viewable – if *true* and this field for given client is not empty, it is shown in the client's chat header along with its current value.

Typical reply:


```
{
    "data": [
        {
            "id": 1,
            "editable": true,
            "viewable": false,
            "name": "Region",
            "type": "input",
            "value": ""
        },
        {
            "id": 2,
            "editable": true,
            "viewable": false,
            "name": "Type",
            "type": "dropdown",
            "value": "Technical,Sales"
        }
    ],
    "status": "ok"
}
```

delete_outbox (GET)

Deletes all outgoing messages that yet have not been sent (for example because of technical failure).

Request:

```
GET /gateway/delete_outbox
```

dialog_states (GET)

Returns a list of available dialog states.

Request:

```
GET /v1/help/dialog_states
```

Typical reply:

```
{"data": [
    "open",
    "closed"
  ],"status": "success"}
```

dialogs (GET)

Returns a list of dialogs. Once an operator takes or is being assigned with a dialog with a client, the messages start to belong to this dialog.

Also this command can be used to get a list of overdue dialogs that were unanswered for specified period of seconds.

Request:

```
GET /v1/dialogs

or

GET /v1/dialogs/<id>

or

GET /v1/dialogs/unanswered?limit=600 – get a list of overdue
dialogs
```

Params:

- <id> – dialog id (optionally). If specified, only info about given dialog will be returned.

- 600 – time in seconds after last client's message after which a dialog is considered overdue.

These filters are supported:

- operator_id
- state (*open* or *closed*)
- order (*asc* or *desc*) – order of records returned. Default is ascending order.

Example of request for dialogs, which belong to operator with operator_id=2:

```
GET /v1/dialogs?operator_id=2
```

Typical reply:


```
{"data": [
     {
            "id": 16490,
            "state": "closed",
            "begin": "2019-06-26T22:20:45 UTC",
            "end": "2019-08-10T15:52:08 UTC",
            "operator_id": 1101,
            "last_message": {
                "id": 23648301,
                "text": "Hi!",
                "coordinates": null,
                "transport": "viber",
                "type": "to_client",
                "read": 1,
                "created": "2019-08-10T14:09:47 UTC",
                "pdf": null,
                "remote_id": null,
                "recipient_status": null,
                "ai_tips": null,
                "dialog_id": 16499,
                "operator_id": 1101,
                "channel_id": 126,
                "attachments": [],
                "photo": null,
                "video": null,
                "audio": null,
                "client_id": 21621
            }
        }
```


Some fields:

- state – dialog state. A dialog may be *open* or *closed*. If a client sends a message to closed dialog, then the client will (by default) receive greetings and any operator will be able to start this dialog. Until a dialog is closed, it is performed by an operator who started it.

  A dialog can be closed because of the following reasons:

  - Time period after last message of the dialog is elapsed. The time period is set up in service settings.
  - The dialog is manually closed by an operator on the web site or with dialogs (PUT).

dialogs (PUT)

Changes a dialog state to *open* or *closed* and sets up its operator with corresponding notification. Also see messages/<id>/transfer (GET)

Request:

```
PUT /v1/dialogs/<id>
```

Params:

- <id> – dialog id (required). See dialogs (GET).

Request body:

```
{"operator_id": 1,
 "state": "open" or "closed"}
```

Typical reply:

```
{
    "status": "success"
}
```

message_types (GET)

Returns a list of available message types.

Request:

```
GET /v1/help/message_types
```

Typical reply:

```
{
```

```
        "data": [
            {
                "id": "system",
                "description": "System message to client"
            },
            {
                "id": "autoreply",
                "description": "Auto message"
            },
            {
                "id": "to_client",
                "description": "Message for client"
            },
            {
                "id": "from_client",
                "description": "Message from client"
            },
            {
                "id": "comment",
                "description": "translation missing:
ru.message_types.comment"
            }
        ],
        "status": "success"
    }
```


messages (GET)

Returns a list of accumulated messages (both from and to clients). If Viber Business, Viber Public or Online chat (widget) transport is used, delivery status is also returned.

Important! Do not use this method too often with high number of records (limit). You will hit API requests limit very soon. Use webhooks instead. Also *start_id* parameter now replaces *offset* parameter to reduce server load.

Request:

```
GET /v1/messages
or
GET /v1/messages/<id>
```

Parameters:

- <id> – message id.

  When this command is used without <id> it returns a list of all accumulated messages. When <id> is specified, the command

returns extended info about specified message, including dialog_id, operator_id and channel_id.

When requesting a list of messages, these filters are supported:

- transport
- channel_id
- client_id
- type (*to_client*, *from_client, autoreply* or *system*)
- dialog_id
- read (read or not read by an operator).
- order (*asc* or *desc*) – order of records returned. Default is ascending order.
- start_date & finish_date – timespan
- operator_id – id of operator
- start_id New – returns array of messages, starting from particular message_id (messages with *id* ⩾ *start_id*)

Example of filtered request:

```
GET
/v1/messages?transport=telegram&type=from_client&read=false
&client_id=101&start_date=09-08-2019&finish_date=11-08-
2019&operator_id=123&start_id=123
```

Some fields in reply:

- coordinates – geo coordinates.
- type:
  - *from_client* – a message from client.
  - *to_client* – a message to client.
  - *system* – system message (like *"Chat was transferred…"*). Such message is not sent to a client.
  - *autoreply* – autoanswer to a client or menu message.

- read – status read or not by operator.
- created – date of message creation (UTC).
- recipient_status – info about delivery status of the message (now applies only to Viber Business/Public, WhatsApp Business API account and Online chat).
- attachments – array of attachments in the message with name and link of each attachment. For backward compatibility some attachment types are additionally returned in separate fields like *photo*, *video* and *audio*.
- insta_comment – true for Instagram comment message;
- extra data – may contain additional info like:
    - external id passed using messages/inbox (POST);
    - post_url - link to post for Instagram comments

Typical reply:


```
{
    "data": {
        "id": 238925990,
        "text": "<p>test test test</p>",
        "coordinates": null,
        "transport": "email",
        "type": "to_client",
        "read": 0,
        "created": "2019-09-23T07:02:46 UTC",
        "pdf": null,
        "remote_id": null,
        "recipient_status": null,
        "ai_tips": null,
        "dialog_id": 9327821,
        "operator_id": 85756,
        "channel_id": 21290,
        "attachments": [],
        "photo": null,
        "video": null,
        "audio": null,
        "insta_comment": true,
        "client_id": 70866186,
        "request_id": 123234486,
        "extra_data":{
          "post_url":"https://www.instagram.com/p/CHCH4cH"
          },
    "status": "success"
}
```

messages (POST)

Sends a message, including buttons, to a client. You can refer a client only by his/her *client id*. So, this client should exist in clients' list. Otherwise, use clients (POST) to create a new client.

Request:

```
POST /v1/messages
```

Request body (without buttons):

```
{"client_id": 7,
  "text": "Hi there!",
  "attachment": "http://chat2desk.com/images/tg_app_en.jpg",
  "pdf": "https://site.com/doc.pdf",
  "type": "to_client",
  "transport": "viber",
  "channel_id": 1,
  "operator_id": 3,
  "open_dialog": false,
  "encrypted": false,
  "external_id": "1122334454"
}
```

Params:

- client_id – client id (see clients (GET)).
- text – message text (required if there's no attachment or PDF).
- attachment – URL for photo attachment. Only direct URLs are supported.
- pdf – URL for PDF attachment. Only direct URLs are supported.

Sending coordinates to a client is not supported.

- type:
  - *to_client* – message to a client (by default).
  - *autoreply* – auto message to a client (like menu reply).
  - *system* – system message (is not sent to the client).
  - *comment* – internal text comment (is not sent to the client).
- transport:

- *whatsapp*
- *wa_cm*
- *wa_infobip*
- *viber*
- *viber_public*
- *viber_business*
- *widget* (online chat)
- *facebook*
- *vkontakte*
- *telegram*
- *email*
- *sms* and more – see transports (GET) for full list.

If omitted, transport of last message from the client will be used.

- channel_id – see channels (GET). If omitted, channel of last message from the client will be used.
- operator_id – see operators (GET). If omitted, the message will be sent from the current dialog's operator. If specified, the message will be mentioned as sent from specified operator.
- open_dialog (*true/false*) – open or not closed dialog when sending message into it. By default - *true*.
- encrypted (*true/false*) New – send the message in encrypted format or not. By default - *false*. The message looks encrypted everywhere inside Chat2Desk but is sent to a client in normal nonencrypted format.
- external_id - if you send message via integration from another messaging platform or CRM-system, you can store id of the message in external system in this field, if it is needed for integration.

Request body (buttons):

```
{"inline_buttons": Button[],
  or
  "keyboard": {
    "buttons": Button[]}}
```

Button array may contain:

```
"Button": {
"type": "reply" // or "location" or "phone" or "email" or
"url"
"text": "Label" // button label
"payload": "My payload" // what the button really sends
(only for "text" type)
"color": "red" // or "green" or "blue" or "yellow" (online
chat only)
"url": "http://..." // button's URL (only for "url" type)}
```

Example:

```
{"keyboard": {"buttons": [{"type":"url", "text": "Site
button", "color": "blue","url": "https://www.google.com"}]}}
```

Parameters supported by messengers:

- reply – VK, Viber Public, Facebook, Telegram, Online chat
- phone – Telegram, Facebook, Viber Public
- email – Facebook
- location – Telegram, Facebook
- url – Viber Public, Online chat
- payload – Viber Public, Facebook, VK, Online chat
- color – Viber Public, VK, Online chat

Inline-buttons support:

- url – Viber Public, Facebook, Telegram
- reply – Viber Public, Facebook, Telegram
- location, phone, email – *none*

We recommend keeping buttons text 38 characters maximum (19 for Cyrillic characters). Otherwise the message might not be sent.

Typical reply:

```
{
    "data": {
        "message_id": 281657474,
        "channel_id": 21288,
        "operator_id": 85756,
        "transport": "telegram",
        "type": "to_client",
        "client_id": 68478143,
        "dialog_id": 9326767,
        "request_id": 32412473
    },
    "status": "success"
}
```

messages/<id>/transfer (GET)

Assigns a message and corresponding dialog to an operator specified. This command should be used to assign a chat which doesn't belong to any operator. Formally, such message doesn't have dialog id.

Request:

```
GET /v1/messages/<id>/transfer?operator_id=<id>
```

Params:

- <id> – message id. See messages (GET).
- operator_id – id of an operator, on whom this message and subsequent dialog will be assigned. See operators (GET).

Typical reply:

```
{
    "data": {},
    "status": "success"
}
```

messages/inbox (POST)

Sends incoming message from client to the system using *external channel*.

Request:

```
POST /v1/messages/inbox
```

Request body:

```
{"channel_id": "1",
    "body": "This is message text",
    "image": <url>,
    "video": <url>,
    "location": "lat lng",
    "extra_data": {"external_id": "111111"},
    "from_client": {
        "id": 123abc or "phone": 19110001122,
        "avatar": <url>,
        "nickname": "API_boy"}}
```

Some parameters:

- external_id – (optional) your id of the message. This id will be sent you back using *inbox* web hook event. This id is also available via messages (GET).
- id – id as you identify a client (any symbols), *except phone number*. If you identify a client with phone number, then use the *phone* parameter.
- phone – client's phone number. Only digits are allowed.

Only one of these two parameters is accepted: *id* or *phone*. If both or none is used, then an error is returned.

- nickname – (optional) client's name.
- avatar – (optional) client's avatar (url to a picture) to display in *Chats* section.

In order this command to work you must have an *external channel* connected to your company. Contact administration.

Typical reply:

```
{
    "status": "success"
}
```

messages/read (GET)

Marks a message as read or unread by operator.

Request:

```
GET /v1/messages/<id>/read
or
GET /v1/messages/<id>/unread
```

Params:

- <id> – message id (required). See messages (GET).

This command marks a message specified as read or unread by operator. Note, that when a message gets answered, the message and the whole dialog are marked as read automatically.

Typical reply:

```
{
    "status": "success"
}
```

operators (GET)

Returns a list of your operators or available operator statuses in the system.

Request:

```
GET /v1/operators

or

GET /v1/operators/statuses
```

When using the *operator* command (without *statuses*), these filters are supported:

- phone
- email
- online (0 or 1). Operator is considered *online* (*online* = 1) if he or she if logged in **in any** status. If an operator manually logs out or the system turns him or her offline because of inactivity, *online* becomes *0*.
- status_id – id of operator's status (see *operators/statuses*).

Typical reply:

```
{
    "data": [
        {
            "id": 85756,
            "email": "example@example.com",
            "phone": null,
            "role": "admin",
            "online": 1,
            "offline_type": null,
            "avatar":"https://storage.chat2desk.com/ava.png",
            "first_name": "Admin",
            "last_name": null,
            "last_visit": "2019-11-25T09:45:44 UTC",
            "external_id": null,
            "opened_dialogs": 0
            "status_id": 1
        }
    ],
    "meta": {
        "total": 1,
        "limit": 20,
        "offset": 0
    },
    "status": "success"
```

```
}
```

When using the *operator/statuses* command, it returns a list of available operators' working statuses (like "Working", "On vacation" etc.) in your chat center.

Typical reply:

```
[
    {
        "id": 0,
        "name": {
            "en": "Working"
        }
    },
    {
        "id": 1,
        "name": {
            "en": "Busy"
        }
    },
    {
        "id": 2,
        "name": {
            "en": "Having break"
        }
    },
    {
        "id": 3,
        "name": {
            "en": "Training"
        }
    },
    {
        "id": 4,
        "name": {
            "en": "Tech. break"
        }
    },
    {
        "id": 5,
        "name": {
            "en": "Vacation"
        }
    }
]
```

operators (PUT)

Changes operator's working status.

Request:

```
PUT /v1/operators/statuses/<operator_id>
```

Request body:

```
{"status_id":2}
```

Params:

- operator_id – id of an operator for whom to change his/her working status. Use operators (GET) for a list of all operators.
- status_id – id of a new status of specified operator. See operators/statuses (GET) for all available statuses in your chat center.

operators_groups (GET)

Returns a list of operator groups in the system.

Request:

```
GET /v1/operators_groups
```

Typical reply:

```
{
    "data": [
        {
            "id": 1632,
            "name": "Technical",
            "operator_ids": [
                90666,
                90667,
                85756
            ]
        },
        {
            "id": 1633,
            "name": "Sales",
            "operator_ids": [
                90667
            ]
        }
    ],
    "status": "success"
}
```

qr-decode (POST)

Decodes QR-code.

Request:

```
POST /v1/qr-decode
```

Request body:

```
{"image_path":
"http://storage.staging.chat2desk.com/clients/inbox/devic
e_1075/2017-client4659e09a3cd6d74.jpg"}
```

Params:

- image_path – path to image with QR-code.

Typical reply:

```
{
    "text": "https://chat2desk.com/"
}
```

regions (GET)

Returns a list of available regions. This list is used for automatic region detection based on client's mobile phone. A list of countries also available – see countries (GET).

Note: This command works only for Russian regions

Request:

```
GET /v1/regions
```

requests (GET)

Returns an info about specified *request* or a list of messages of specified *request*.

> *Request* — is a set of messages within a dialog with a client. As a rule, request starts with first client message and finishes when the dialog is closed. When the client continues to chat in closed dialog, a new request starts in the same dialog.

Command:

```
GET /v1/requests/<id>
or
GET /v1/requests/<id>/messages
```

Params:

- id – request id.

When requesting an info about specified request, use *requests/<id>*.
Example of typical reply includes dialog id and request's tags:


```
 "id": 8164246,
     "dialog_id": 4126193,
     "tags": [
         {   "id": 2251,
             "label": "Sales",
             "description": "Client sales request",
             "group_id": 579,
             "group_name": "Tag group 1"}]
```


When requesting a list of request messages, use *requests/<id>/messages*.

Typical reply:


```
[
    {
        "id": 286627344,
        "text": "QWERTY",
        "video": null,
        "photo": null,
        "coordinates": null,
        "transport": "telegram",
        "type": "in",
        "read": 1,
        "created": 1574752570,
        "clientID": 68478143,
        "operatorID": 85756,
        "channelID": 21288,
        "companyID": 74647,
        "dialogID": 9326767,
        "conversationID": null,
        "is_menu": null,
        "audio": null,
        "pdf": null,
        "extra_type": "normal",
        "tricolor": null,
        "remote_id": null,
        "recipient_status": null,
        "gateway_status": "pending",
        "extra_data": null,
        "job_id": null,
        "sent_at": null,
        "ai_tips": null
    },
    {
        "id": 286627709,
        "text": "hello!",
        "video": null,
        "photo": null,
        "coordinates": null,
```

```
            "transport": "telegram",
            "type": "out",
            "read": 0,
            "created": 1574752595,
            "clientID": 68478143,
            "operatorID": 85756,
            "channelID": 21288,
            "companyID": 74647,
            "dialogID": 9326767,
            "conversationID": null,
            "is_menu": null,
            "audio": null,
            "pdf": null,
            "extra_type": "normal",
            "tricolor": null,
            "remote_id": null,
            "recipient_status": null,
            "gateway_status": "sent",
            "extra_data": {},
            "job_id": "e8ba68992046e75a1558d88e",
            "sent_at": "2019-11-26T07:16:35.000Z",
            "ai_tips": null
        }
    ]
```


requests/close (PUT)

Closes specified *request*.

> *Request* — is a set of messages within a dialog with a client. As a rule, request starts with first client message and finishes when the dialog is closed. When the client continues to chat in closed dialog, a new request starts in the same dialog.

Command:

```
PUT /v1/requests/close
```

Request body (without buttons):

```
{"request_id": 537136
  or
 "client_id": 37817}
```

Params:

- request_id
- client_id – id of a client. Current request of specified client will be closed.

Typical reply:

```
{
    "state": "closed"
}
```

roles (GET)

Returns a list of available operator roles.

Request:

```
GET /v1/help/roles
```

Typical reply:

```
{"data": [
    "unconfirmed",
    "operator",
    "supervisor",
    "admin",
    "disabled",
    "deleted"
  ],"status": "success"}
```

Some possible values:

- unconfirmed – a user hasn't confirmed his\her e-mail yet.
- supervisor – an operator with extended rights (supervisor).
- disabled – a blocked operator.
- deleted – a deleted operator.

scenarios (GET)

Returns a list of available self-service menu items.

Request:

```
GET /v1/scenarios/menu_items
```

Params:

- channel_id – (optional) id of a channel. Self-service menu is different for each channel.
- level – (optional) level of the menu items. First (root) level is 1.

Some fields in the results:

- command – quick command to call this menu item from the web interface.
- parentID – id of parent menu item. If the current item is root item, *null* is returned.
- position – position of current menu item at it's level.
- level – level of current menu item.
- url – url to call when this menu item is chosen by a client.
- url_enabled – if equals *1*, then the system will send back to the client the text response that above url returns.
- assign_id – if equals *1*, then the chat with current customer will be assigned to specified operator (if not assigned yet).
- assign_subject – to whom to assign the chat (see prev. item).
- image – link to image or PDF which is sent when this menu item is chosen (as well as text).
- tag_id – tag to assign to the *request* (not the client) when this menu item is chosen.

Typical reply:


```
{
    "data": [
        {
            "id": 39477,
            "name": "End this chat",
            "command": "End",
            "text": "Thank you and see you soon!",
            "parentID": null,
            "channelID": 21288,
            "position": 1,
            "level": 1,
            "url": null,
            "url_enabled": 0,
            "assign_id": null,
            "assign_subject": null,
            "image": null,
            "tag_id": null,
            "smart_method_name": null
        },
        {
            "id": 40102,
            "name": "Option 1",
            "command": "1",
            "text": "You chosed option 1",
            "parentID": null,
            "channelID": 21288,
            "position": 2,
            "level": 1,
```

```
                "url": "",
                "url_enabled": 0,
                "assign_id": null,
                "assign_subject": "",
                "image": "",
                "tag_id": null,
                "smart_method_name": ""
            },
            {
                "id": 40103,
                "name": "Option 2",
                "command": "2",
                "text": "%hello, %client! You chosed option 2.",
                "parentID": null,
                "channelID": 21288,
                "position": 3,
                "level": 1,
                "url": "",
                "url_enabled": 0,
                "assign_id": 1632,
                "assign_subject": " OperatorsGroup",
                "image": "",
                "tag_id": 17700,
                "smart_method_name": ""
            }
        ],
        "status": "success"
    }
```


scenarios (POST)

Sends self-service menu item to a client.

Request:

```
POST /v1/scenarios/send_menu_item
```

Params:

- client_id – id of a client to whom to send specified menu item.
- menu_item_id – (optional) menu item id to send to specified client. If omitted, root menu item from last client message channel is sent.

Typical reply:


```
{
    "data": {
        "message_id": 281621970,
        "channel_id": 21288,
        "operator_id": 85756,
        "transport": "telegram",
        "type": "to_client",
        "client_id": 68478143,
        "dialog_id": 9326767,
        "request_id": 32409446,
```

```
            "menu_item_id": "40103"
        },
        "status": "success"
    }
```

statistics (GET)

Returns aggregated statistics data, for example operators' reaction and reply times, number of (new) clients and requests per day, operators activity, ratings, tags of requests and more. See statistics description here: in English, in Russian.

Important! You must have "Aggregated statistics" feature turned on for your company. If it's not, contact us.

Request:

```
GET /v1/statistics/report=operator_replies&date=2019-06-20
```

Params:

- report_name – name of 5 available statistic reports (without quotes):
    - operator_replies
    - request_stats
    - operator_stats
    - operator_events
    - rating

  See aggregated statistics description above.
- date – day (in your company's time zone) for which statistics will be returned. You can specify 1 day only. Keep in mind, that statistics for current day will be calculated only tomorrow at around 2 a.m.

The statistics is calculated daily for 2 previous days starting from the day of enabling for your company. If there's no calculated statistics data for specified day, empty values will be returned.

Typical reply:

```
{
    "data": [
        {
            "client_phone": "[tg] 90651782",
            "request_start": 1574752570,
            "client_region": "",
```

```
                "client_extra_2": "",
                "request_start_mode": "offline",
                "request_starter": "other",
                "request_time": 548,
                "working_request_time": 548,
                "first_operator_role": "admin",
                "request_tags": "bad",
                "working_reaction_time": 25,
                "reply_start": 1574752595,
                "channel_id": 21288,
                "client_country": "",
                "client_info": "{\"1\":\"St.Petersburg\"}",
                "ban_status": "",
                "client_assigned_name": "",
                "incoming_menu_messages": 0,
                "incoming_messages": 1,
                "outgoing_menu_messages": 0,
                "first_operator_group1": "Technical",
                "first_operator_name": "Admin",
                "outgoing_messages": 1,
                "client_tags": "",
                "operators_in_request": 1,
                "request_start_t": "23:16:10",
                "client_extra_1": "",
                "first_operator_group2": "",
                "channel_name": "Main",
                "request_id": 32990906,
                "client_comment": "",
                "client_name": "Mr. Smith",
                "client_extra_3": "",
                "reaction_time": 25,
                "request_terminator": "",
                "reply_start_d": "25.11.2019",
                "first_operator_id": 85756,
                "request_start_d": "25.11.2019",
                "client_id": 68478143,
                "client_messenger": "telegram",
                "client_is_new": "false",
                "reply_start_t": "23:16:35"
            }
        ],
        "meta": {
            "total": 1,
            "limit": 20,
            "offset": 0
        }
    }
```


tag_groups (POST)

Creates a group of tags.

Request:

```
    POST /v1/tag_groups
```

Request body:

```
{"tag_group_name": "Name of the group"
  "order_show": 5,
  "display": 0 or 1}
```

Params:

- tag_group_name – name of the group.
- order_show – is used to sort tags in UI.
- display – display (*1*) or not (*0*) this tag group in UI when manually editing a client or request.

Typical reply:

```
{
    "data": {
        "id": 3608,
        "group_name": "Tags group 1",
        "order_show": 1,
        "display": 1
    },
    "status": "success"
}
```

tags (DELETE)

Deletes a tag from a client or a request.

Request:

```
DELETE /v1/tags/<id>/delete_from
```

Params:

- <id> – id of a tag to delete.

Request body:

```
{"client_id": 1020}
or
{"request_id": 627172}
```

Params:

- client_id – id of a client from whom you want to delete the specified tag.
- request_id – id of a client from whom you want to delete the specified tag.

Typical reply:

```
{
    "status": "success"
}
```

tags (GET)

Returns a list of your tags.

Request:

```
GET /v1/tags/<id>
```

Params:

- <id> – id of a tag (optional). If omitted, the command returns full list of tags.

Typical reply:

```
{
    "data": [
        {
            "id": 18731,
            "group_id": 3280,
            "group_name": "quality",
            "label": "bad",
            "description": "bad"
        },
        {
            "id": 19136,
            "group_id": 3280,
            "group_name": "quality",
            "label": "good",
            "description": "good"
        },
        {
            "id": 17700,
            "group_id": 3607,
            "group_name": "tags group 1",
            "label": "vip",
            "description": "vip client"
        }
    ],
    "meta": {
        "total": 3,
        "limit": 20,
        "offset": 0
    },
    "status": "success"
}
```

tags (POST)

Creates a new tag.

Request:

```
POST /v1/tags
```

Request body:

```
{"tag_group_id": 1,
  "tag_label": "ABC1",
  "tag_description": "My tag description",
  "tag_bg_color": "5b9ee6",
  "tag_text_color": "ffffff",
  "order_show": 5}
```

Params:

- tag_group_id – id of tag group inside which the new tag will be created. Every tag should belong to a group.
- tag_label – tag label (ticket) — 20 chars max.
- tag_description – tag description.
- tag_bg_color – tag's background color number.
- tag_text_color – tag's foreground color number.
- order_show – is used to sort tags in UI.

Typical reply:

```
{
    "data": {
        "id": 20013,
        "group_id": 3607,
        "group_name": "tags group 1",
        "label": "ABC1",
        "description": "My tag description"
    },
    "status": "success"
}
```

tags/assign_to (POST)

Assigns tags to a client or a request.

Request:

```
POST /v1/tags/assign_to
```

Request body:

```
{
"tag_ids": [1,2],
 "assignee_type": "client" or "request",
 "assignee_id": "1020"
}
```

Params:

- tag_ids – list of tags ids.
- assignee_type – to what you want to assign the tag: to client or to request.
- assignee_id – id of a client or request to which you want to assign specified tags.

Typical reply:

```
{
    "status": "success"
}
```

templates (GET)

Returns your list of templates.

Request:

```
GET /v1/templates/<id>
```

Params:

- <id> – (optional) id of a template. If omitted, full list of your templates is returned.

Typical reply:

```
{
    "data": [
        {
            "id": 26675,
            "command": "1",
            "text": "%hello, %client! :)",
            "name": "Hello, client!",
            "created": "2019-11-06T10:40:51 UTC"
        },
        {
            "id": 26697,
```

```
                "command": "2",
                "text": "This is API manual",
                "name": "API pdf",
                "file":
"https://storage.chat2desk.com/companies/company_74647/templates/
template-08-27-51-538e1985e45a4ac4da40.pdf",
                "created": "2019-11-07T06:43:15 UTC"
            }
        ],
        "meta": {
            "total": 2,
            "limit": 20,
            "offset": 0
        },
        "status": "success"
    }
```


transfer_to_group (GET)

Arranges a dialog on group of operators.

Request:

```
    GET
/v1/messages/<message_id>/transfer_to_group?group_id=<group_id>
```

Upon completion, this command returns an info about the operator of specified group that the dialog was assign to.

Typical reply:


```
    {
        "data": {
            "assignee": {
                "id": 85756,
                "email": "example@example.com"
            }
        },
        "status": "success"
    }
```


transports (GET)

Returns a list of available transports and their supported attachments.

Request:

```
    GET /v1/help/transports
```

Typical reply:


```
    {"data": {
        "whatsapp": {
          "from_client": {
```

```
            "text": "yes",
            "image": "yes",
            "audio": "no",
            "pdf": "no",
            "video": "yes",
            "location": "yes"
          },
          "to_client": {
            "text": "yes",
            "image": "yes",
            "audio": "yes",
            "pdf": "yes",
            "video": "no",
            "location": "yes"
          }"status": "success"}
```


web_analytics_data (GET)

Returns a list of URLs and UTM marks of clients' chats along with Google, Yandex, Comagic and Roistat user ids. Useful for marketing purposes.

To obtain this data, you must have *URL/UTM tracking* feature turned on. See *Settings* > *Web analytics* section.

Also, your clients must start chats using our widget with messengers, see *Settings* > *Widget*.

Request:

```
GET /v1/ web_analytics_data?date_from=2019-09-
01&date_to=2019-09-30&order=asc
```

Params (optional):

- <date_from> yyyy-mm-dd
- <date_to> yyyy-mm-dd
- <order> asc or desc
- <client_id> unique client id in our system, see clients (GET)

Typical reply:


```
{
    "data": [
        {
            "message_id": 288195145,
            "client_id": 84537162,
            "url": "http://yoursite.html",
            "click_time": "2019-11-27T08:51:01.000Z",
```

```
                "utm": "",
                "google_id": null,
                "yandex_id": null,
                "comagic_id": null,
                "utm_source": null,
                "utm_medium": null,
                "utm_content": null,
                "utm_term": null,
                "utm_custom": null,
                "roistat_visit": null
            }
        ],
        "meta": {
            "total": 13,
            "limit": 1,
            "offset": 0
        }
    }
```


webhooks (DELETE)

Deletes specified web hook.

Request:

```
DELETE /v1/webhooks/<id>
```

Params:

- <id> – web hook id that you want to delete – see webhooks (POST).

Typical reply:


```
{
    "status": "success"
}
```


webhooks (GET)

Returns a list of web hooks with their events.

Request:

```
GET /v1/webhooks/<id>
```

Params:

- <id> – (optional) web hook id – see webhooks (POST). If omitted, a list of all web hooks is returned.

Typical reply:

```
{
    "data": [
        {
            "id": 5518,
            "name": "My first web hook",
            "url": "https://yoursite.com/",
            "events": [
                "inbox"
            ],
           "status": "enable",
          "errors": null
        }
    ],
    "status": "success"
}
```

webhooks (POST)

Creates a new web hook for various events.

Important! We resend unsuccessful webhook event 3 times: in 10 seconds, in 5 minutes and in 2 hours. In order to avoid overload, **webhook will be disabled,** if during 5 hours no respond of successful data acceptance was received. You can use method webhooks (GET) and look at section “errors” for more details.

Request:

```
POST /v1/webhooks
```

Request body:

```
{"url": "https://yoursite.com/",
 "name": "name",
 "events": ["inbox", "outbox", "new_client",
 "add_tag_to_client", "add_tag_to_request",
 "delete_tag_from_client", "close_dialog",
 "close_request", "dialog_transferred", "new_qr_code"]}
```

Params:

- name – name of created web hook. Up to 5 web hooks are allowed.
- url – URL on your server where the requests will come. Any URL is allowed but we recommend https. If empty value (*null*) is set then web hook is deleted.
- events – events list for which web hook will be triggered.

- *inbox* – incoming message from a client.

  Example of data:
  Array ([message_id] => 106773169 [type] => from_client [text] => Text [transport] => telegram [client_id] => 100 [operator_id] => [dialog_id] => [channel_id] => 17 [photo] => [coordinates] => [audio] => [pdf] => [client] => Array ([id] => 17340089 [phone] => [tg] 807892 [client_phone] => [name] => Peter [assigned_name] => [external_id] =>) [hook_type] => inbox [request_id] => 8725057 [attachments] => Array ( ) [is_new_request] => 1 [is_new_client] =>[extra_data] => Array ( ) [event_time] => 2019-07-28T23:01:40Z)

- *outbox* – outgoing message from system to a client.

  Example of data:
  Array ([message_id] => 106773577 [type] => to_client [text] => Bc[[transport] => telegram [client_id] => 17 [operator_id] => 41164 [dialog_id] => 4324035 [channel_id] => 17404 [photo] => [coordinates] => [audio] => [pdf] => [client] => Array ([id] => 17340089 [phone] => [tg] 807892 [client_phone] => [name] => Peter [assigned_name] => [external_id] =>) [hook_type] => outbox [request_id] => 8725095 [attachments] => Array ( ) [is_new_request] => [is_new_client] => [event_time] => 2019-07-28T23:01:40Z)

- *new_client* – first incoming message from new client.

  Example of data:
  Array ([id] => 18360561 [phone] => [tg] 1938691 [name] => Peter [avatar] => https://storage.chat2desk.com/clients/avatars/2018-05/03-client2242811-5aeb3098bf2ec.jpg [assigned_name] => [comment] => [extra_comment_1] => [client_phone] => [extra_comment_2] => [extra_comment_3] => [channels] => Array ([id] => 17404 [transports] => Array ([0] => telegram)) [region_id] => [country_id] => [custom_fields] => [hook_type] => new_client [event_time] => 2019-07-28T23:01:40Z)

- *add_tag_to_client* – a tag is assigned to a client.

  Example of data:
  Array ([id] => 4835 [name] => TestTag [description] => This is test tag [group_id] => 1190 [group_name] => Tag test group [client_id] => 17340089 => [all_client_channels] => 21288 => [hook_type] => add_tag_to_client [event_time] => 2019-07-28T23:01:40Z)

- *add_tag_to_request* – a tag is assigned to a request.

  Example of data:
  Array ([id] => 4835 [name] => TestTag [description] => This is test tag [group_id] => 1190 [group_name] => Tag test group [client_id]

=> 17340089 [hook_type] => add_tag_to_request [event_time] => 2019-07-28T23:01:40Z)

- *delete_tag_from_client* – a tag is deleted from a client.

  Example of data:
  Array ([id] => 4835 [name] => TestTag [description] => This is test tag [group_id] => 1190 [group_name] => Tag test group [client_id] => 17340089 [hook_type] => delete_tag_to_client [event_time] => 2019-07-28T23:01:40Z)

- *delete_tag_from_request* New - a tag is deleted from a request.

  Example of data:
  Array ([id] => 4835 [name] => TestTag [description] => This is test tag [group_id] => 1190 [group_name] => Tag test group [client_id] => 17340089 [hook_type] => delete_tag_from_request [event_time] => 2019-07-28T23:01:40Z)

- *client_updated* New - info in client card is changed.

  Example of data:
  Array ([id] => 18360561 [phone] => [tg] 1938691 [name] => Peter [avatar] => https://storage.chat2desk.com/clients/avatars/ava.jpg [assigned_name] => [comment] => [extra_comment_1] => [client_phone] => [extra_comment_2] => [extra_comment_3] => [channels] => Array ([id] => 17404 [transports] => Array ([0] => telegram)) [region_id] => [country_id] => [custom_fields] => [hook_type] => client_updated [event_time] => 2019-07-28T23:01:40Z)

- *close_dialog* – a dialog is closed.

  Example of data:
  Array ([dialog_id] => 4324035 [request_id] => 8725057 [created] => 1540282584 [updated] => 1540464524 [client_id] => 1000 [channel_id] => 17404 [operator_id] => 41164 [transport] => telegram [hook_type] => close_dialog [event_time] => 2019-07-28T23:01:40Z)

- *close_request* – a request is closed.

  Example of data:
  Array ([dialog_id] => 3951761 [request_id] => 8086095 [created] => 1550671015 [updated] => 1550671015 [channel_id] => 126 [client_id] => 27129636 [operator_id] => 39538 [transport] => telegram [hook_type] => close_request [event_time] => 2019-07-28T23:01:40Z)

- *dialog_transferred* – a dialog is assigned to an operator or assigned operator is changed

  Example of data:
  Array ([dialog_id] => 3951761 [request_id] => 8086095 [created] => 1550671015 [updated] => 1550671015 [channel_id] => 126 [client_id] => 27129636 [operator_id] => 39538 [current_operator_id] => 39538 [transport] => telegram [last_operator_id] => 8236 [scenario_id] => 124 [hook_type] => dialog_transferred [event_time] => 2019-07-28T23:01:40Z)

- *new_qr_code* – new QR code to restore WhatsApp connection appeared on site in *Settings/Accounts*. It means your WhatsApp connection got broken and needs repair.

  Example of data:
  Array ([qrcode_data] => 1@OxrhCyoevcoool+z5GK+sBFL14vtAPH72jAl9eKQKO8VrSMSW3bsWzhx,Cf25AuQqLZmicHDNH9x3Og6gDMyXR9uVF16888S74g8=,BDVBTbHLr9gYB3LW52dNQQ== [qrcode_image] => ...= [phone] => 339002538473 [device_owner] => service [hook_type] => new_qr_code [event_time] => 2019-07-28T23:01:40Z)

ai_hints_requested – this web hook is for AI (Artificial Intelligence) provider and used in 2 ways:

1. An operator or system requested AI hints (suggestions) in prompter mode. Hints (suggestions) from AI provider are expected in return. To understand the context the provider should have all messages using web hooks *inbox* and *outbox* events (recommended) or messages (GET).

   Example of data:
   {"client_id": 11910, "operator_id": 1, "dialog_id": 331, "request_id": 2343, "channel_id": 1, "message_id": 8205 [event_time] => 2019-07-28T23:01:40Z}

   In return, AI provider should send hints (suggestions) in the following format:

   ```
   "message_id": 8205,
   "hints": [{"id": 1, "text": "answer 1", "url": "", "relevance": 90,
   "link": ""},{"id": 2, "text": "answer 2", "url": "", "relevance": 97,
   "link": ""}]
   ```

2. An operator has chosen one AI hint from the list above. Used to teach AI engine on correct answers.

   Example of data:
   {"client_id": 11910, "message_id": 8205, "hint_id": 2, "hook_type": "ai_hints_requested" [event_time] => 2019-07-28T23:01:40Z}

To modify existing web hook use webhooks (PUT). See webhooks (GET) for your current web hooks.

Old API command web_hook (POST) operates with one default web hook called *__default* and may soon become depreciated.

Testing your web hooks

To test your webhoook you can use this site:

https://webhook.site/

Important! Do not forget to change your web hook URL after testing.

web_hook (POST)

Important! This method will be depreciated soon. Use webhooks (POST) instead for multiple web hooks.

Sets up URL for web hook which is used for receiving json-formatted info on various events.

Request:

```
POST /v1/companies/web_hook
```

Request body:

```
{"url": "https://yoursite.com/"
"events": ["inbox", "outbox", "new_client", "new_tag",
"add_tag_to_client", "delete_tag_from_client",
"close_dialog", "new_qr_code"]}
```

You can check your current web hook with api_info (GET).

- url – URL on your server where our requests will come. Any URL is allowed but we recommend https. If empty value (null) is set then web hook is deleted.

- events – events list on which web hook will be triggered.
    - *inbox* – incoming message from a client.
    - *outbox* – outgoing message from system to a client.
    - *new_client* – first incoming message from new client.
    - *new_tag* – new tag is added to the system.
    - *add_tag_to_client* – a tag is assigned to a client.
    - *delete_tag_from_client* – a tag is deleted from a client.
    - *close_dialog* – a dialog is closed.
    - *new_qr_code* – new QR code to restore WhatsApp connection appeared on site in *Settings/Accounts*. It means your WhatsApp connection got broken.

webhooks (PUT)

Updates specified web hook.

Request:

```
PUT /v1/webhooks/<id>
```

Params:

- <id> – web hook id that you want to update – see webhooks (POST).
- optional parameter "status": enable/disable – use this parameter if you want to activate or deactivate existing wehhook. If omitted, webhook status will not be changed.

For other parameters – see webhooks (POST).

Typical reply:

```
{
    "data": {
        "id": 5518,
        "name": "New name",
        "url": "https://yoursite.com/",
        "events": [
```

```
                "inbox"
            ],
            "status": "enable",
            "errors": null
        },
        "status": "success"
    }
```