API manual

This API is designed to work with Chat2Desk service.

LIST OF CHANGES

Date	Version	Changes
1.11.2018	1.27	*new_tag* web hook event is discontinued and removed from this manual.
26.10.2018	1.26	Added examples of web hook 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 clients (GET).
28.08.2018	1.22	• Parameter *nickname* added for clients (POST) • New method added – companies (PUT)
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, Live 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. Use this link to access a collection of preset API commands for Chat2Desk in **Postman app**:

   https://documenter.getpostman.com/view/262638/RWaPskvM

   You must input your **API token** to use these commands (see item 5 below).

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

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

6. Commands that return large list of data support pagination using URL parameters:

   a. *limit* – number of records returned (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*

7. In general, the system is designed to answer those clients who first contacted your company and 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. Contact these newly created clients reasonably because your WhatsApp- or Viber-account might get blocked for spam in case of too many contacts or messages.

8. There are 2 ways of obtaining messages from your clients:
    - Handling a new message event with webhooks (POST). This method is recommended.
    - Downloading of accumulated messages with messages (GET).

BEFORE YOU BEGIN

1. Obtain your <*token*> on Chat2Desk site in *Settings* > *API*. Also make sure there that your API-access level is at least to *demo*. This information is also available with api_modes (GET).
2. There are two ways to work with messages:
    a. Your clients use regular messengers to connect to the system. In this case use webhooks (POST) or messages (GET). To send use clients (POST) and messages (POST).
    b. 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 to your clients – you should use web hooks.

LIST OF API COMMANDS

- api_info (GET)

  Returns current API version, your API access level, number of API requests per month and more.

- messages (GET)

  Returns a list of accumulated messages (both from and to clients).

- requests (GET)

  Returns a list of messages for specified requests.

- messages (POST)

  Sends a message to a client.

- webhooks (POST)

  Creates a new web hook for various events.

- webhooks (GET)

  Returns a list of web hooks or info about specified web hook.

- webhooks (PUT)

  Updates specified web hook.

- webhooks (DELETE)

  Deletes specified web hook.

- messages/inbox (POST)

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

- messages/<id>/transfer (GET)

  Assigns a message and corresponding dialog to an operator.

- clients (GET)

  Returns a list of clients with their info.

- clients (POST)

  Creates a new client so you can contact this client afterwards.

- clients (PUT)

  Assigns a name to client.

- companies (PUT)

Stores company's custom data.

- operators (GET)

  Returns a list of operators in the system.

- operators_groups (GET)

  Returns a list of operator's groups in the system.

- transfer_to_group (GET)

  Transfers (arranges) a dialog on group of operators.

- dialogs (GET)

  Returns a list of dialogs.

- dialogs (PUT)

  Changes a dialog state and sets up its operator.

- tags (GET)

  Returns a list of tags.

- tags (POST)

  Creates a new tag.

- tag_groups (POST)

  Creates a new tag group.

- tags/assign_to (POST)

  Assigns one or more tags to a client.

- tags (DELETE)

  Deletes client tag.

- templates (GET)

  Returns a list of templates.

- custom_client_fields (GET)

  Returns a list of custom fields of client info card.

- menu_items (GET) New!

  Returns a list of available self-service menu items.

- send_menu_item (POST) New! New!

  Sends self-service menu item to a client.

- qr-decode (POST)

  Decodes QR-code.

- roles (GET)

  Returns a list of available roles.

- dialog_states (GET)

  Returns a list of available dialog states.

- clients (PUT)

  Assigns a name to a client.

- channels (GET)

  Returns a list of channels with info.

- regions (GET)

  Returns a list of available regions.

- countries (GET)

  Returns a list of available countries.

- transports (GET)

  Returns a list of available transports and their supported attachments.

- messages/read (GET)

  Tags a message as read or unread by operator.

- api_modes (GET)

  Returns a list of available API access levels.

ADDITIONAL COMMANDS

- channels/<id>/clients (GET)

  Returns a list of clients by specified channel.

- questions/<id>/send (GET)

  Sends specified menu item to the client when using scripts (see *Help* > *Scripts manual*).

- clients/<id>/questions (GET)

Returns a list of menu items that were sent to a client during specified date span (optionally) when using scripts (see *Help* > *Scripts manual*). Request:

```
GET /v1/clients/<id>/questions?start_date=
10-09-2011&finish_date=10-09-2019
```

- delete_outbox (GET)

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

  Request:

  ```
  GET /gateway/delete_outbox
  ```

- web_hook (POST)

  This command sets up one URL for web hook event on new message from a client, new outgoing message, adding/deleting a client's tag, on creating a new tag, closing a dialog and more. This method will be **depreciated** soon. Use webhooks (POST) instead for multiple web hooks.

api_info (GET)

Returns:

- Current API version used by our service.
- Your current API access level – see api_modes (GET).
- Number of API requests of your company for current month.
- Number of available channels
- Custom info (see companies (PUT)) and other parameters.

Request:

```
GET /v1/companies/api_info
Host: api.chat2desk.com
Authorization: <token>
```

Params:

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

Typical reply:

```
{"data": {
    "mode": "demo",
    "hook": null,
    "current_version": 1,
    "requests_this_month": 98,
    "channels": 2,
    "company_name": "Test company"
  },"status": "success"}
```

messages (GET)

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

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 given 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)

Example of filtered request:

```
GET
/v1/messages?transport=telegram&type=from_client&read=false
&client_id=101
```

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 and Online chat transport).

- 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*.

requests (GET)

Returns 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, the dialog is opened and a new request starts.

Command:

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

Params:

- id – request id.

messages (POST)

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

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}
```

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).
- transport:
  - *whatsapp*
  - *viber*
  - *viber_public*
  - *viber_business*
  - *widget*
  - *facebook*
  - *telegram*
  - *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 other than current operator of the dialog, the dialog will be reassigned to specified operator.
- open_dialog (*true/false*) – open or not closed dialog when sending message into it. By default - *true*.

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"
"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
- phone – Telegram, Facebook
- email – Facebook
- location – Telegram, Facebook
- url – Viber Public
- payload – Viber Public, Facebook, VK
- color – Viber Public, VK

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.

webhooks (POST)

Creates a new web hook for various events.

Request:

```
POST /v1/webhooks
```

Request body:

```
{"url": "https://yoursite.com/",
 "name": "name",
  "events": ["inbox", "outbox", "new_client",
  "add_tag_to_client", "delete_tag_from_client",
  "close_dialog", "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] =>)
    - *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] =>)

- *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)

- *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 [hook_type] => add_tag_to_client)

- *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)

- *close_dialog* – a dialog is closed.

  Example of data:
  Array ([dialog_id] => 4324035 [request_id] => 8725057 [created] => 1540282584 [updated] => 1540464524 [channel_id] => 17404 [operator_id] => 41164 [transport] => telegram [hook_type] => close_dialog)

- *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)

To modify existing web hook use web_hook (PUT). See web_hook (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

We have dedicated URL to test web hooks:

https://web.chat2desk.com/site/check-hook

Set your web hook to this URL, open it in browser and see the data it receives. Remember, that this URL is used by other testers and there may be their's data. Do not forget to change your web hook URL after testing.

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.

webhooks (PUT)

Updates specified web hook.

Request:

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

Params:

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

For other parameters – see webhooks (POST).

webhooks (DELETE)

Deletes specified web hook.

Request:

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

Params:

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

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"
    "from_client": {
        "id": 123abc or "phone": 19110001122,
        "nickname": "API_boy"}}
```

Some parameters:

- 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.

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

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 clients (PUT).

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
```

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.
- 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.

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 via WhatsApp or Viber 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.
- 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 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.

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 turn this option on for you.

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".

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 api_info (GET).

operators (GET)

Returns a list of your operators.

Request:

```
GET /v1/operators
```

These filters are supported:

- phone
- email
- online

Typical reply:

```
{"data": [
    {
      "id": 1,
      "email": "sss@gmail.com",
      "phone": null,
      "role": "admin",
      "online": false,
      "offline_type": null,
      "first_name": null,
      "last_name": null,
      "last_visit": "2016-03-11T19:45:22 UTC"
    },
    {
      "id": 2,
      "email": "111@11.com",
      "phone": "15550001122",
      "role": "supervisor",
      "online": false,
      "first_name": "John",
      "last_name": "Smith",
      "last_visit": "2016-02-25T17:25:43 UTC"
    }
  ],
  "meta": {
    "total": 2,
    "limit": 20,
```

```
        "offset": 0
      },
      "status": "success"}
```

operators_groups (GET)

Returns a list of operator groups in the system.

Request:

```
    GET /v1/operators_groups
```

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.

dialogs (GET)

Returns a list of your dialogs. Once an operator starts a chat with a client, the messages start to belong to a 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*)

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

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

Typical reply without specifying dialog id:

```
{"data": [
    {
      "id": 1,
      "state": "closed",
      "begin": "2015-07-22T20:52:39 UTC",
      "end": "2015-07-26T20:05:41 UTC",
      "operator_id": 1
    },
    {
      "id": 2,
      "state": "open",
      "begin": "2015-07-22T20:34:24 UTC",
      "end": null,
      "operator_id": 1
    },
    {
      "id": 3,
      "state": "open",
      "begin": "2015-07-24T11:30:32 UTC",
      "end": null,
      "operator_id": 1
    },
    …
  ],
  "meta": {
    "total": 429,
    "limit": 20,
    "offset": 0
  },
  "status": "success"}
```

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 an auto answer 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"}
```

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.

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) — 8 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.

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.

tags/assign_to (POST)

Assigns one or more tags to a client.

Request:

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

Request body:

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

Params:

- tag_ids – list of tags ids.
- assignee_type – to whom you want to assign tag: to client or to request. Now only client assignment is supported — *client*.
- assignee_id – id of a client or request to which you want to assign specified tags. Now only client assignment is supported.

tags (DELETE)

Deletes a tag from a client.

Request:

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

Params:

- <id> – id of a tag to delete.

Request body:

```
{"client_id": 1020}
```

Params:

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

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.

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.

menu_items (GET) New!

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.

send_menu_item (POST) New!

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.

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.

roles (GET)

Returns a list of available 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.

dialog_states (GET)

Returns a list of available dialog states.

Request:

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

Typical reply:

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

channels (GET)

Returns a list of channels with info.

Request:

```
GET /v1/channels
```

This filter is supported:

- phone

Typical request with the filter:

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

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).

Request:

```
GET /v1/regions
```

countries (GET)

Returns a list of available countries in the system.

Request:

```
GET /v1/countries
```

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"}
```

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.

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
```

Typical reply:

```
{"data": [
    {
      "name": "disabled",
      "description": "API disabled"
    },
    {
      "name": "demo",
      "description": "1000 requests per month"
```

```
    },
    {
      "name": "normal",
      "description": "50000 requests per month"
    },
    {
      "name": "unlimited",
      "description": "unlimited requests per month"
    }
  ],"status": "success"}
```

web_hook (POST)

This method may 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.