Chat2Desk

Python Scripts Manual

chat2desk.com

Changelog

21.04.2022

New API description (now in Postman).

15.12.2021

Rate limit is applied to *send_message* command.

11.02.2021

moderate_message command is depreciated.

22.12.2020

2 new script events:

- *delete_tag_from_request*
- *add_tag_to_request*

Earlier

Earlier changelog is not provided.

Intro

Python scripts can be used to add custom logic to messages flow inside your chat center. Scripts are stored on our server and can be edited by you online.

See our commercial scripts list here: in English, in Russian.

Apart script commands, you can use our API (see API manual).

Typical use cases

Here are some examples of what can be done with the scripts. At the end of this document there are examples of actual scripts.

- Dialogs routing

  Assign dialogs with a client to your operators based on phrases, customer's tags, current time, operators' busyness, etc.

- Check the delay of service actions

  Rearrange chats that were delayed by operators' fault to other online operators or supervisors.

- Send info to CRM

  Send client and dialog info to CRM system via CRM API.

- Assign tags to client and requests
- Upon tag assignment different scenarios in these scripts or ***Sales Tunnels*** can be started.

  > ***A 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 text in a closed dialog, the dialog is opened and a new request starts.

How to start

To start using scripts ask administration to turn this feature on for you. This is paid feature.

1. As admin go to **Settings > Scripts**.

2. Make sure to turn on **script logging** to receive script error messages: go to the bottom of the **Scripts** section and find *Logging into your messenger* section. You must have your client id to receive the logs into your messenger.

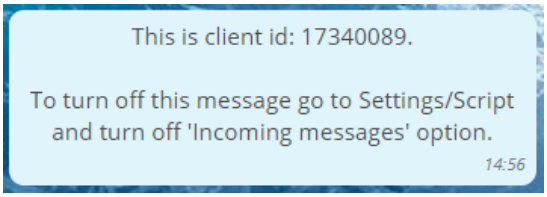
   To get your client id:

   - Turn on *Incoming message* event on the top of **Settings > Scripts** section. Make sure there's this code in the script's *new_message_handler*. It should be there by default. If not, copy-paste it.

     ```
     class Handler:
     def new_message_handler(self, input_data, c2d):
     c2d.send_message(input_data['client']['id'], "This is client id: " + str(input_data['client']['id']) + ".\n\nTo turn off this message go to Settings/Script and turn off 'Incoming messages' option.", "system")
     ```

   This script sends client id as system message to every chat with an incoming message. Send a message to any messenger connected to your Chat2Desk account.

   - Check the chat with yourself in **All chats** section. There will be system message like this:

3. Put this client id in *Logging into your messenger* section to receive script debug messages.
4. Check *events* that you want to trigger your script.
5. Modify the script: edit event handlers that correspond to the events that you've checked above.
6. Click **Save**.
7. Test your script with test input data:
   - In *Test input data* check the input data and click an event you want to test at the top of this section.
   - Click *Run on test input_data* to test your script.
   - See output info in *Output* field.

Note. You can use script to manage chat assignment between operators, it is advised to turn off all chat assignment options in **Settings > General > Self-service menu** and Chatbots to avoid chat arrangement conflicts.

Inside Python scripts feel free to use our API commands like this:

```
results = requests.put( 'https://api.chat2desk.com/v1/dialogs/'+str(did), params = {'state':'closed', 'operator_id':5000}, headers={'Authorization': api_token })
```

Events that trigger the script

There are 14 events. You can turn these events on and off in **Settings > Scripts**.

New message received

new_message_handler

Occurs after a new message is received by your chat center. Such data comes inside *input_data*:

```
{
"message": {
"id": 111,
"is_menu": 1010,
"dialogID": 12,
"operatorID": 121,
"text": "message",
"transport": "whatsapp",
"photo": "https://site.com/images/users/client/48-5741232.jpg",
"video": URL,
"audio": URL,
"pdf": URL,
"coordinates": "55.32165 55.43401"
},
"client": {
"id": 1112,
"phone": "375447697415",
```

```
"name": "John Connor",
"assigned_name": "The one"
},
"channel": {
"id": 1112,
"phone": "13757777777",
"name": "first channel"
}
}
```

This event happens **before** auto answer message and self-service menu message. If you want to block them, return 'not send menu':

```
def new_message_handler(self, input_data, c2d):
...
return('not send menu')
```

Before sending message

before_sending_message_handler

Occurs when an operator sends a message, just before the message is sent.

Before closing dialog

before_closing_dialog_handler

Occurs when a dialog is closed (both manually and automatically), just before it is actually closed.

After closing dialog

after_closing_dialog_handler

Occurs after a dialog is closed (both manually and automatically).

Every 60 seconds (auto checking)

auto_checking_handler

This event occurs every 60 seconds. Such data comes inside ***input_data***:

```
"time": current time
```

After successful QR code recognition

qr_code_result_handler

Occurs when a QR code in incoming message was recognized. Option to recognize QR codes in incoming messages should be enabled for your company first — contact us.

A call from external system

manually_handler

Your company has a special URL (web hook), that can be called from external services. Any info can be passed to this URL in JSON format using GET or POST. When this URL is called, ***manually_handler*** is executed with the info passed via ***input_data***. Use this event to integrate our scripts with events in external services like telephony and CRM.

See your URL in **Settings > Script > Web hook call from external service**.

Chatbot didn't trigger on incoming message

chat_bot_not_triggered_handler

Occurs when an incoming message came and your chatbot (see **Settings > Chatbot**) didn't trigger.

Chat transfered from one operator to another

dialog_transfer_handler

Occurs when a chat is transferred from one operator to another. Works on the web site only.

Request from new client

new_request_handlers

Occurs when a new request is started. Do not confuse a request with a message.

Client info changed

client_updated_handler

Occurs when a client info card is closed using ***Ok*** button on the site. This event doesn't happen on client tags assignment.

Phone number requested

request_phone_handler

Occurs when an operator clicks ***Request phone number*** button in **Chats** section. You can send ***Request phone*** through your own script. It can be turned on for Facebook, Viber public and Telegram transports. This option should be enabled for your company first — contact us.

Tag assigned to request

add_tag_to_request_handler

Occurs when a tag is added to request manually by an operator, using self-service menu or using a chatbot.

Tag deleted from request

delete_tag_from_request_handler

Occurs when tag is deleted from request manually by an operator, using self-service menu or using chatbot.

Script commands

Here's the list of Chat2Desk Python scripts commands.

send_message

Sends text message to a client.

```
c2d.send_message(client_id, text, type, file, channel_id)
```

> This command is restricted by a **rate limit**: the system allows to send **no more than 15 messages per second**. If the limit is exceeded, the following error will be returned: "429 Too many requests".

Parameters:

client_id is an id of a client to whom you want to send a message (int). See above how to get client id of yourself for tests.

text is a text to send (string).

type (optional) is a message type. Possible values: ***to_client*** (default), ***autoreply*** or ***system***. Use ***autoreply*** to send an automatic reply without assigning chat to any operator. Use ***system*** to send system message to chat with a client without actually sending it to the client.

file (optional) is a direct URL to picture or PDF file.

channel_id (optional) is an id of a channel if the client has dialogs in more than one. If omitted, last client message channel will be used.

send_question

Sends self-service menu item to a client. Menu is created on the site in **Settings > Self-service menu** section.

```
c2d.send_question(client_id, question_id)
```

Parameter:

question_id — as shown in **Settings > Self-service menu** section (integer).

get_client_info

Returns a client info: name, assigned name, comment, timestamp of the first and the last message, phone number or id, country (by the phone number), region (by the phone number), last transport used, channel, avatar and more.

```
c2d.get_client_info(client_id)
```

get_operators

Returns a list of all your operators accounts in the system: first and last name, number of open dialogs, timestamp of last visit, login (e-mail), phone, role, online status, offline status and more.

```
c2d.get_operators()
```

get_client_dialogs

Returns a client's current dialog info. Remember, that if a dialog of a client is in **New chats** section, the client's message doesn't have an operator's and dialog's ids until the dialog is assigned to any operator.

```
c2d.get_client_dialogs(client_id)
```

Also, this command helps to determine the client's last operator to assign the dialog to this operator that last served this client.

get_online_operators

Returns a list of online operators as well as the same info as for *get_operators*.

```
c2d.get_online_operators(true or false)
```

Parameter:

Only with new chats available (True or *False)* — if *False* or omitted, a list of all online operators is returned. If *True* then only online operators with **New chats** section available to them are returned. See **Settings > Security & Access rights** section.

get_questions

Returns an array of self-service menu items requested by a client during a specified timespan. This command is useful for retrieving the client's context in self-service menu hierarchy.

```
c2d.get_questions(client_id, '10-10-2021', '10-12-2022')
```

get_last_question

Returns last menu item (id, text and image) sent to a client — same info as for *get_questions*.

```
c2d.get_last_question(client_id)
```

get_unanswered_dialogs

Returns a list of dialogs that have unanswered message from a client and respective client list. Useful to transfer "stuck" dialogs to free operator.

```
c2d.get_unanswered_dialogs(limit)
```

Parameter:

limit (sec) — time period after which a dialog is considered as unanswered.

get_new_messages

Returns a list of new messages (new chats). It can be used to check new chats and assign them to operators.

```
c2d.get_new_messages()
```

transfer_dialog

Transfers (arranges) a dialog to an operator. This command should be used when a dialog already has one operator and you want to transfer it to another.

```
c2d.transfer_dialog(dialog_id, operator_id, 'Take this chat please')
```

If the dialog does not have an operator (it is in **New chats** section), you have to use *transfer_message* command.

transfer_message

Transfers (arranges) a new dialog with specified message to an operator. This command should be used when a dialog doesn't have an operator (in **New chats** section).

```
c2d.transfer_message(message_id, operator_id, 'Take this chat please')
```

transfer_message_to_group

Transfers (arranges) a dialog with specified message to a group of operators. The operator is chosen as set up by admin in **Settings > Operator > Groups** on the web site.

```
c2d.transfer_message_to_group(message_id, group_id)
```

To get operator groups ids use *get_operators_groups()*.

get_operators_groups

Returns a list of operator groups. To get a list of groups of one specified operator use *get_operator_group_ids()*.

```
c2d.get_operators_groups()
```

get_operator_group_ids

Returns a list of one specified operator groups ids.

```
c2d.get_operator_group_ids(operator_id)
```

get_company_info

Returns current company info such as: name, work schedule, current work-mode (***online*** or ***offline***) and more.

```
c2d.get_company_info()
```

get_last_message_id

Returns id of last message in a dialog with a client.

```
c2d.get_last_message_id(dialog_id, message type, timespan)
```

Parameter:

message type is a type of a message: 1 — from client, 2 — from operator, 3 — autoreply, 4 — system message.

timespan is a timespan in seconds to check for the last message. It can be used to exclude old dialogs.

send_template

Sends specified template to a client. See **Settings > Templates** section.

```
c2d.send_template('abc', client id)
```

Parameter:

'abc' **—** template's quick command.

get_menu_items

Returns a list of self-service menu items.

```
c2d.get_menu_items(channel id, menu items level)
```

Parameters:

channel id (optional), see ***channels GET API*** function or **Settings > Accounts** section. If omitted, menu items from all channels will be returned.

menu items level (optional). First (root) level is 1 (menu item level starts from 1, unlike API). If omitted, all menu item levels will be returned.

send_menu_item

Sends self-service menu item to a client.

```
c2d.send_menu_item(client id, menu item id)
```

Menu item level starts from 1 (unlike API).

Parameters:

client id, obtain it from ***input_data***.

menu item id (optional) if omitted, root menu from last user message channel will be sent.

Script examples

Below are examples of ready-made scripts. Please contact us If you want to improve your Chat2Desk experience or change the way some functions work. We will create a script for you. See more information about scripts on our website.

“Send WhatsApp/Viber visit card after client’s phone call”

When a client performs a voice call, the telephony system should call the URL (web hook), specified for your company — see ***manually_handler*** above and pass a calling party phone number and, optionally, ***calling event*** in JSON format.

Here’s a script which checks that this client has already contacted your chat center and if not, creates this client. Then, the script sends a message (visit card) to this client. The message considers current time.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
import re, sys, time, json
from datetime import datetime
from requests import get, put, post, delete

# This script sends text into WhatsApp when called via POST request from external system. The request should be made to URL specified above in “Web hook call from external service (manually_hander)” (it should be turned on). Phone field in the request is obligatory. If a name is also specified, the new chat with client will be named with this name. Phone should consist only digits. To avoid ban for spam, new clients contacts per day is restricted using limitHere variable. For this script to work Chat2Desk’s “Write first” feature should be on

class Handler:
reload(sys)
sys.setdefaultencoding('UTF8')

def new_message_handler(self, input_data, c2d):
return

def manually_handler(self, input_data, c2d):
#print input_data
kind = 'to_client'
name=''

# 3 different texts for work time, offwork time and weekend days
msg=['Hi (work time) ',\
'Hi (offwork time) ',\
'Hi (weekend)']
chat_state = True # Close or not the chat after sending message
today=datetime.today()
if today.weekday() in (5,6):
number = 2
else:
```

```
number = 0 if today.hour in range(9,19) else 1 # working hours

text=msg[number]
if 'phone' in input_data and input_data['phone']:
phone = str(input_data['phone'])
else:
return 'No phone'
if 'name' in input_data: name = input_data['name']
if phone[0] == '+': phone=phone[1:] # Removing +

api_headers['Authorization']=c2d.token

data=get(api_url % 'clients?phone='+phone,headers=api_headers)
if not data: return ''
client=json.loads(data.text)
if client['meta']['total']==0:
if not counter_control(): return 'Too many new clients for today (not sent)'

post(api_url % 'clients?transport=whatsapp&phone='+phone,headers=api_headers)
data=get(api_url % 'clients?phone='+phone,headers=api_headers)
if not data: return ''
client=json.loads(data.text)

if not client['data'] or len(client['data'])==0:
return ''
if name != '':
put(api_url % 'clients/' + str(client['data'][0]['id']),headers=api_headers,params={"nick-
name":name}) # Renaming

return sendText(c2d, client['data'][0]['id'],text,kind,chat_state)

def counter_control():
fldHere='extra_comment_2'
```

```
limitHere=500                # limit of a new clients per day
data=get(api_url % 'clients?limit=1',headers=api_headers)
if not data: return False
id=json.loads(data.text)['data'][0]['id']
content=json.loads(data.text)['data'][0][fldHere]
cnt=1;
curDay=int(time.strftime("%d"))
if content:
value=json.loads(content)
if curDay==int(value['date']):
if int(value["count"])>=limitHere: return False
else: cnt=int(value["count"])+1
data=json.dumps({"date":curDay,"count":cnt})
data={"extra_comment_2": data}
put(api_url % 'clients/%s' % str(id),headers=api_headers,data=data)
return True

# Send a message
def sendText(c2d,clientID, text, kind='autoreply', close_chat=True):
info = c2d.send_message(str(clientID), text, kind)

if close_chat and "message_id" in info:
headers={'Authorization':c2d.token}
mn=str( info["message_id"] )
obj=get(api_url %'messages/'+mn,headers=headers).json()
if not obj or not obj["data"] or not obj["data"]["dialog_id"]: return 'Cannot close dialog'
try:
dg = str( obj["data"]["dialog_id"] )
except:
return 'Cannot close dialog'
oID = obj["data"]["operator_id"]
put(api_url % 'dialogs/'+dg,headers=headers,params={"operator_id":oID, "state":"closed"})
return ''
```

```
# Common functions
api_headers={'Authorization':''}
api_url='https://api.chat2desk.com/v1/%s'
```

“Don’t disturb operators while a client uses self-service menu”

Do not notify the operators when a client uses self-service menu. Once the client writes something else — arrange the dialog to least busy online operator. Done by arranging a dialog to dumb operator whose last name is ***Bot***.

```
def new_message_handler(self, input_data, c2d):
# operators - list of all operators
operators = c2d.get_operators()
bot_operator = None
# find operator with last_name = 'bot'
for operator in operators:
if operator['last_name'] == 'bot':
bot_operator = operator
# if bot operator was found, continue logic
if bot_operator:
# if message is new, transfer message to bot
if not input_data['message']['dialogID']:
c2d.transfer_message(input_data['message']['id'], bot_operator['id'])
# if message belongs to bot operator, check: client message is menu item (or not)
else:
if int(input_data['message']['operatorID']) == int(bot_operator['id']):
# if message is not menu item, transfer to free online operator
if not self.is_menu_items(input_data['message']['text']):
# if we found free online operator, transfer message to him
free_operator = self.find_free_online_operator(c2d)
if free_operator:
c2d.transfer_message(input_data['message']['id'], free_operator['id'])
```

```
# If message is menu item, return True
# Note! We've added new field in message data: is_menu. If it's Null, then this isn't menu item. Otherwise it is equal to menu item's command. This old example doesn't use this new field.
def is_menu_items(self, message):
# list of all menu items
menu_items = ['0', '1', '2', '3', '00', 'End']
# compare client message with menu items
for item in menu_items:
if item == message:
return True
return False

# Return free operator or None
def find_free_online_operator(self, c2d):
# operator - list of online operators
operators = c2d.get_online_operators()
# if all operators are offline, skip logic
if len(operators) > 0:
free_operator = operators[0]
# find operator with minimum number of opened dialogs
for operator in operators:
if operator['opened_dialogs'] < free_operator['opened_dialogs']:
free_operator = operator
return free_operator
else:
return None
```