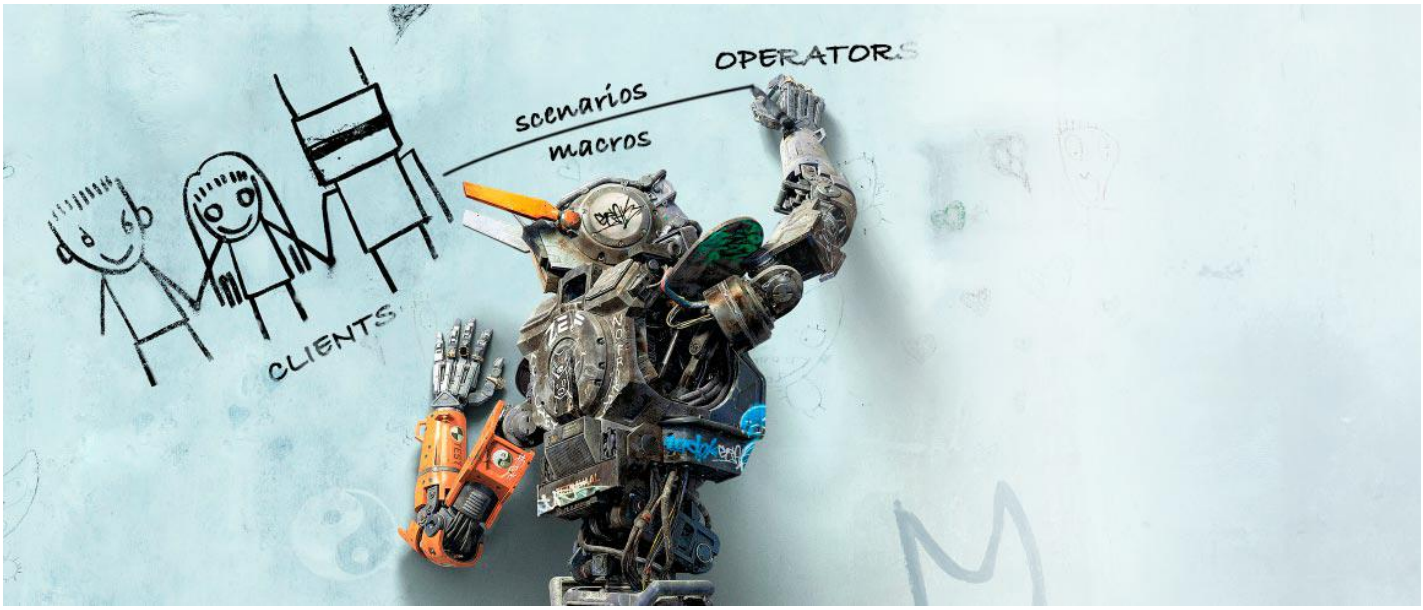

Scripts manual

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

Apart script commands, you can use our API. See API manual in *Help* section of the site.

Date	Version	Changes
25.10.2018	1.01	New commands added: • get_menu_items • send_menu_item
28.08.2018	1.00	New command added – send_template

Typical use cases

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

1. Dialogs routing

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

2. Message filtering

   Filter incoming and outgoing messages and even moderate which outgoing messages are OK and which are not.

3. Check the delay of service actions

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

How to start

1. To start using scripts ask administration to turn this feature on for you. This is paid feature.
2. As admin go to *Settings* > *Scripts*.
3. Check *events* that you want to trigger your script.
4. Modify the script: edit event handlers that correspond to the events that you checked above.
5. Click *Save*.

We strongly advice to turn on logging at the bottom of *Scripts* section to receive error and logging messages in your messenger.

Testing your script with test input data

1. In *Test input data* check the input data and click an event you want to test.
2. Click *Run* to test your script.
3. See output info in *Output* field.

Note, that if you 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 *Chat bots* to avoid 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 9 events. You can turn these events on and off in *Settings* > *Scripts*.

1. New message received

new_message_handler

Occurs after a new message is received by your chat-center. Such data comes inside *handler->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* autoanswer 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')
```

2. Before sending message

before_sending_message_handler

Occurs before a new message is sent by an operator. Such data comes inside *handler->input_data*:

```
{
  "message": {
    "id": 111,
    "is_menu": None,
    "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": "3757777777",
    "name": "first channel"
  },
  "operator": {
    "id": 1112,
    "role": "supervisor"
  }
}
```


3. Before closing dialog

before_closing_dialog_handler

Occurs ***before*** a dialog is closed: both manually or automatically. Such data comes inside *handler->input_data*:


```
{
  "dialog_id": 111,
  "client": {
    "id": 1112,
    "phone": "1375447697415",
    "name": "John Connor",
    "assigned_name": "The one"
  }
}
```


4. After closing dialog

after_closing_dialog_handler

Occurs *after* a dialog is closed: both manually or automatically.

5. Every 60 seconds (auto checking)

auto_checking_handler

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

```
{
  "time": current time
}
```

6. After successful QR-code recognition

qr_code_result_handler

Occurs when a QR-code in incoming message was recognized.

7. After call from external system

manually_handler

Your company has a special URL (web hook), that can be called from external service and some 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*.

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

8. Chat bot didn't trigger on incoming message

chat_bot_not_triggered_handler

Occurs when your chat bot (*Settings/Chat bot*) didn't trigger on incoming message.

9. Chat transfer 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.

10. New client request

new_request_handler(self, input_data, c2d)

Occurs when a new request is started. Do not confuse new request with new *message*.
Request — is a set of messages within a dialog with a client. As a rule, request starts with the first client message and finishes when the dialog is closed. When the client continues to text in closed dialog, the

dialog is opened and a new request is started.

11. Client info changed

client_updated_handler(self, input_data, c2d)

Occurs when a client info popup is closed with Ok button on the site. This event doesn't happen on client tags.

Script commands

Here's the list of Python commands that we currently have:

1. send_message

Sends text message to a client.

```
c2d.send_message(94212, 'test!', 'autoreply')
```

Parameters (in order shown):

- *client_id* – id of a client to whom you want to send a message. Obtain the id using API or from input data like this:

  ```
  c2d.send_message(input_data['client']['id'], 'Hi!', 'autoreply')
  ```

- *text to send*
- *type (optional)* – 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.

2. send_question

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

```
c2d.send_question(94212, 4321)
```

Parameters (in order shown):

- *client_id* – id of a client.
- *question_id* – as shown in *Settings* > *Self-service menu* section.

3. 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
- Channel
- Avatar

```
c2d.get_client_info(94212)
```

Parameter:

- *client_id* – id of a client.

4. get_operators

Returns a list of all operators:

- First and last name
- *Number of open dialogs*
- Timestamp of last visit
- Login (e-mail)
- Phone
- Role
- Online status
- Offline status

```
c2d.get_operators()
```

5. 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 not have an operator's and dialog's ids until the dialog is assigned to any operator.

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

```
c2d.get_client_dialogs(94212)
```

Parameter:

- *client_id* – id of a client.

6. get_online_operators

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

```
c2d.get_online_operators(True)
```

Parameter:

- *Only with new chats available* – (True/False). If ***False*** or omitted list of all online operators is returned. If ***True*** only online operators with new chats available to them are returned.

7. get_questions

Returns an array of menu items requested by a client. This command is useful for retrieving the context for given user.

This info is returned:

- Id
- Text
- Image
- Timestamp of creation

```
c2d.get_questions(5369, '10-10-2015', '10-12-2017')
```

Parameters (in order shown):

- *client_id* – id of a client.
- *start date*
- *end date*

8. 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(5369)
```

Parameter:

- *client_id* – id of a client.

If current menu position is root, then *menu item id* = 0 is returned.

9. 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. Info returned – ids of such dialogs and ids of clients.

```
c2d.get_unanswered_dialogs(18000)
```

Parameter:

- *limit (sec)* – time period after which a dialog is considered unanswered.

10. 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()
```

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

```
c2d.transfer_dialog(81984, 1899, 'Take this chat please')
```

Parameters (in order shown):

- *dialog_id*
- *operator_id*
- *note to operator when transferring the chat*

12. 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(81984, 1899, 'Take this chat please')
```

Parameters (in order shown):

- *message_id*
- *operator_id*
- *note to operator when transferring the chat (optional)*

13. 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 > Operators in groups* on the web site.

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

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

14. get_operators_groups

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

```
c2d.get_operators_groups()
```

15. get_operator_group_ids

Returns a list of given operator groups ids.

```
c2d.get_operator_group_ids(500)
```

Parameter:

- *operator_id*

16. moderate_message

Sends a message for moderation to an operator. The operator should accept or decline the message by clicking on flag near the message. The dialog is transferred to moderating operator.

```
c2d.moderate_message(81984, 1899)
```

Parameters (in order shown):

- *message_id*
- *operator_id* – moderating operator

17. get_company_info

Returns current company info such as:

- name
- work schedule
- current work-mode (online or offline) and more

```
c2d.get_company_info()
```

18. get_last_message_id

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

```
c2d.get_last_message_id(100, 2, 2*24*60*60)
```

Parameters (in order shown):

- *dialog_id*
- *message type:* (1 — from client, 2 — from operator, 3 — autoreply, 4 — system message)
- *timespan:* timespan in seconds to check for the last message. It can be used to cut off old dialogs.

19. send_template

Sends specified template to a client.

```
c2d.send_template('abc',100)
```

Parameters (in order shown):

- *'abc'* – template's quick command (see *Settings/Templates*).
- *100* – client id.

20. get_menu_items

Returns a list of menu items. There's a corresponding API command (see API manual).

```
c2d.get_menu_items(169,3)
```

Parameters (in order shown):

- *169* – (optional) channel id (see *channels GET* API function or *Settings/Accounts*). If omitted, menu items from all channels will be returned.
- *3* – (optional) menu items level. First (root) level is 0. If omitted, all menu item levels will be returned.

21. send_menu_item

Sends menu item to a client. There's a corresponding API command (see API manual).

```
c2d.send_menu_item(720283,201)
```

Parameters (in order shown):

- *720283* – client id (obtain it from *input_data*).
- *201* – (optional) menu item id. If omitted, root menu from last user message channel will be sent.

Scripts examples

1. Send WhatsApp/Viber visit card after client 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 already contacted your chat center and if not, creates this client. Then, the script sends a message (visit card) to this client. The message takes into account 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={"nickname":name}) # Renaming

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

def counter_control():
    fldHere='extra_comment_2'
    limitHere=50                # 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'
```

2. Arrange dialogs based on content of the message

When a client writes into closed dialog or this client is a new client, check the message and if it contains words like “working”, “fail”, “problem”, “issue” and “help” – arrange this dialog to specified operator who is in charge for urgent technical support.

```
def new_message_handler(self, input_data, c2d):
    reload(sys)
    sys.setdefaultencoding('UTF8')
    # check that this message doesn't belong to existing dialog
    if not input_data['message']['dialogID']:
      if self.find_spec_words(input_data['message']['text']):
          c2d.transfer_message(input_data['message']['id'], 100)

  def find_spec_words(self, text):
    spec_words = ['working', 'fail', 'problem', 'issue', 'help']
    for word in spec_words:
      if word in text.lower():
        return True
    return False
```

2. Check outgoing message for abuse or credit card number

Check that outgoing message from an operator doesn’t contain abuse or digits that look like credit card number. If any condition is met – sends that message for moderation.

```
def before_sending_message_handler(self, input_data, c2d):
  if not self.valid_by_censure(input_data['message']['text']):
    c2d.moderate_message(input_data['message']['id'], 1899)
  if not self.valid_by_number(input_data['message']['text']):
    c2d.moderate_message(input_data['message']['id'], 1899)

def valid_by_censure(self, text):
  black_words = ['fuck', 'dick']
  for word in black_words:
    if text in word:
      return False
  return True

def valid_by_number(self, text):
  return not re.search('[0-9]{13,16}', text)
```

3. Don’t disturb operators while a client uses self-service menu

Do not notify the operators until 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 - *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
```

4. In case of delay of service transfer the dialog to another operator

If a client's message has been unanswered for more than 10 mins, then transfer a dialog to another online supervisor.

```
def auto_checking_handler(self, input_data, c2d):
    # find unanswered dialogs (10 min)
    dialogs = c2d.get_unanswered_dialogs(60*10)
    # if unanswered dialogs exists, do logic
    if len(dialogs) > 0:
      # find list of online operators
      operators = c2d.get_online_operators()
      # find operator with role supervisor
      for operator in operators:
      # find supervisor exists, transfer all unanswered dialogs to him
        if operator['role'] == 'supervisor':
          for dialog_id in dialogs:
            c2d.transfer_dialog(dialog_id, operator['id'])
```

5. Arrange dialogs to operators based on two last letters in their names

This script is useful for distributing clients of different branches between corresponding operators.

```
def new_message_handler(self, input_data, c2d):
    # Check, inbox message belongs to dialog. If message without dialog, execute transferring logic
    if not input_data['message']['dialogID']:
      # operator - list of online operators
      operators = c2d.get_online_operators()
      # assigned_name - client assigned name
      assigned_name = input_data['client']['assigned_name']
      # if assigned name is empty, we don't have region data in client name and skip logic
      if assigned_name:
        # code - region data is key for filter operators
        code = assigned_name[-2:]
        # find relevant operator
        relevant_operator = next((operator for operator in operators if operator['last_name'][-2:] == code), None)
        # if relevant operator exists, transfer message to him
        if relevant_operator:
          c2d.transfer_message(input_data['message']['id'], relevant_operator['id'])
          return
      # if relevant_operator does not exists, transfer message to first online operator
      if len(operators) > 0:
        c2d.transfer_message(input_data['message']['id'], operators[0]['id'], "Online operators for " + str(code) + "
not found. Transferred to you as default operator.")
```

6. Operators utilisation

Arrange new chats to online operator who has the fewest number of chats.

```
def new_message_handler(self, input_data, c2d):
    # Check, that inbox message belongs to a dialog. If message is without a dialog, execute transfer logic
    if not input_data['message']['dialogID']:
      # 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
        # transfer message to free operator
        c2d.transfer_message(input_data['message']['id'], free_operator['id'])
```

7. Arrange clients between clinics based on their choice

When a new client contacts, the client is asked to choose what clinic this client belongs to. The client sends a number and then is transferred to corresponding clinic's operator. If this client had already chosen a clinic before – the script assigns this client to corresponding operator without asking.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
import re, sys
class Handler:
  def new_message_handler(self, input_data, c2d):
    reload(sys)
    # get current work-mode
    online_status = c2d.get_company_info()['online']
    sys.setdefaultencoding('UTF8')
    if not input_data['message']['dialogID']:
      # find previous dialogs, if found, transfer client to previous operator
      dialogs = c2d.get_client_dialogs(input_data['client']['id'])
      if len(dialogs) > 0:
        c2d.transfer_message(input_data['message']['id'], dialogs[0]['operator_id'])
        c2d.send_message(input_data['client']['id'], 'Please wait for reply…', 'autoreply')
        return
      # try to transfer message according to message text
      else:
        client_message = input_data['message']['text']
        if client_message == str(1):
          c2d.transfer_message(input_data['message']['id'], 1000)
          # if offline - send another text
          if online_status == False:
            c2d.send_message(input_data['client']['id'], ‘Clinic 1 is offline and will answer you ASAP…’, 'autoreply')
          else:
            c2d.send_message(input_data['client']['id'], 'You have chosen Clinic 1. Please wait for reply…',
'autoreply')
          return
      # same conditions for other clinics
      c2d.send_message(input_data['client']['id'], "Please select your clinic:\
\n1 – Clinic1\
\n2 – Clinic 2\
\n3 – Clinic 3", 'autoreply')
```