**Documentation**

**EdgeServapi – Device Automation Platform**

# About

EdgeServapi Platform consists of two components:

1. Server-side software (SERVAPI):
   1. SERVAPI webserver (REST API)
   2. Admin portal
   3. Device registration portal (for future)
2. Client-side software (Edge):
   1. EDGE Client (HTTP client – Python program)
   2. EDGE Program (custom program collecting metrics from sensors or doing soothing else)

# Client-side software (EDGE)

## Edge Client

This is http client written in Python3.

Files of Edge Client:

<client>.py -> client software written in Python3. Can be converted to executable.

settings.txt -> text file with settings

clientcert.pem -> client certificate

clientkey.pem -> client key pair file

ca-cert.pem -> public key of server CA (which signed SERVAPI certificate)

out-> directory

out/queue.txt -> text file containing list of files for transfer from agent to SERVAPI DB

autoprovision.sh (for future -> automatic device provisioning and registration)

get\_device\_id.py -> script to get device identifier (for device provisioning)

### Communication

Client-server communication is HTTPS (TLSv2)-based. Client repeatedly sends HELLO requests (interval is defined in settings) and the server responds with action: standby (no action), update (need to download files from server and execute command) or command (need to execute commands and send report to the server).

#### Client messages

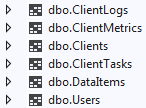
URI (query):

1. /hello -> client GET message to the server (including health metrics) to retireve tasks
2. /update -> client GET message to the server to download files
3. /command -> client POST message to send output of execute commands
4. /upload -> client POST message to send file to the server
5. /confirm -> client GET message to confirm that new settings were applied
6. /report -> client GET message to report a failure

Client – Server communications algorithm is displayed below.



### Database structure



Clients -> all Edge Client data (primary key=sn what is device identifier based on S/N or MAC).

ClientTasks -> all tasks: update (push files to client and execute command), command (execute commands), settings (change client settings), upload (pull file from client).

ClientLogs -> all logs.

ClientMetrics -> all client metrics from hello messages.

DataItems -> all files that are linked to tasks (push, pull or task result output).

Users -> Admin Portal user data (usernames and hashed passwords).

**Currently we use MySQL.**

### SERVAPI backend code listing

SERVAPI and Admin Portal products are based on Microsoft ASP .NET Core 3.1.

#### Data format for messages exchange between clients and server.

In most of cases data classes are serialized and converted to JSON string to transfer to client/server.

Below are details of classes that describe client settings, client request structure and server responses structures.

##### Client settings description

public class Client //EDGE Client Description

{

public string sn = "";

public string mac = "";

public string model = "";

public string hostname = "";

public string location = "";

public int customer\_id;

public string servapi\_host;

public string servapi\_port;

public string provisioned;

public string ssh\_enabled;

public string ssh\_command;

public string client\_update\_freq; //shoule have crontab format like "0 0 0 0 0 python /root/client.py"

public string program\_update\_freq; //shoule have crontab format like "0 0 0 0 0 ./root/program"

public string client\_version;

public string program\_dir;

//More fields to be added

}

##### Standard client request description

public class StandardRequest //Standard HTTP request parameters from EDGE

{

public string sn = "";

public string mac = "";

public string model = "";

public string hostname = "";

public string location = "";

public int customer\_id = 0;

public long uptime; // Uptime in sec

public string ip = "";

public double memory; //Memory utilization, %

public double cpu; //CPU load instant, %

public double free\_disk; //Free size in / MB

public string time = "";

public string id; //Используется для сопоставления реквестов отправки команды/апдейта с ответом клиента

public string custom\_field = ""; //used for reporting faults and future

public string filename; // For program data upload

public string filetype; // For program data upload

public double network; //reserved

}

Server responses to clients

public class Response\_Command

{

public string action = ""; //command (list of commands) update or standby (do nothing)

public List<string> payload = new List<string>();

public string id; //request id -> to figure out folder

}

public class Response\_Update

{

public string action = ""; //command (list of commands) update or standby (do nothing)

public string location = "";

public string command = "";

public List<long> files\_ind = new List<long>();

public List<string> files = new List<string>();

public string id; //request id -> to figure out folder

}

public class Response\_Standby

{

readonly public string action = "standby"; //command (list of commands) update or standby (do nothing)

}

public class Response\_Settings

{

public string action = ""; //command > settings

public string servapi\_host;

public string servapi\_port;

public string provisioned;

public string ssh\_enabled;

public string ssh\_command;

public string client\_update\_freq; //shoule have crontab format like "0 0 0 0 0 python /root/client.py"

public string program\_update\_freq; //shoule have crontab format like "0 0 0 0 0 ./root/program"

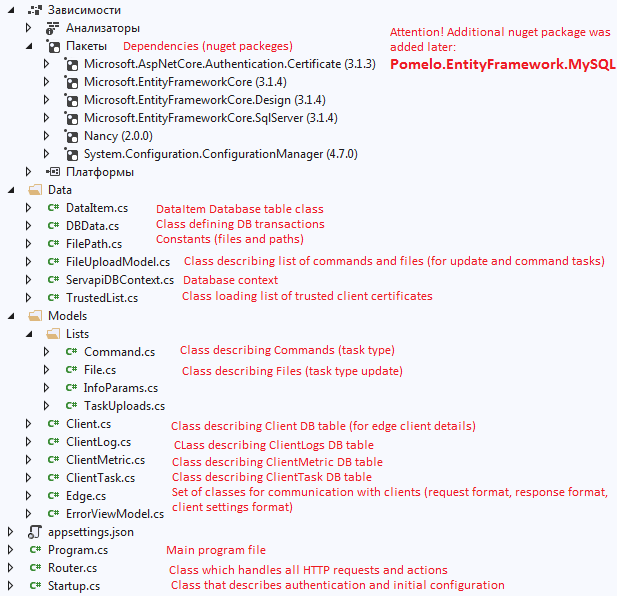
public string client\_version;

public string program\_dir;

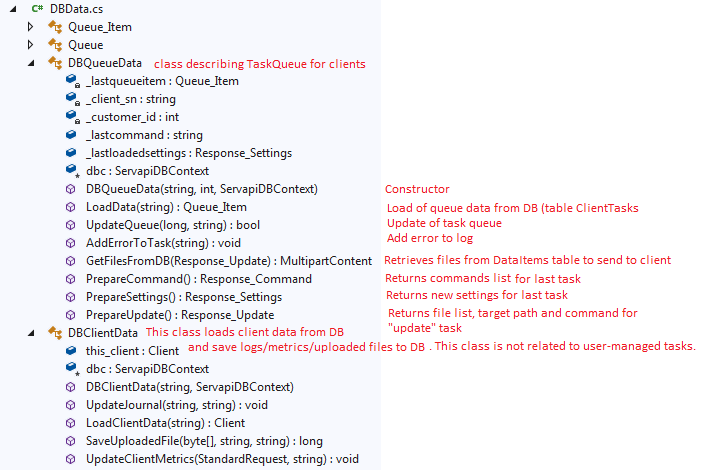
public string id; //request id -> to figure out folder

}

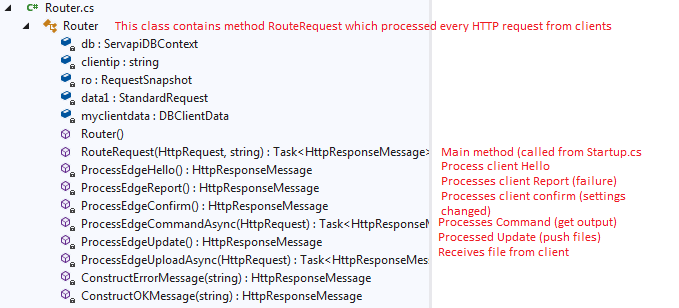
## SERVAPI project file structure



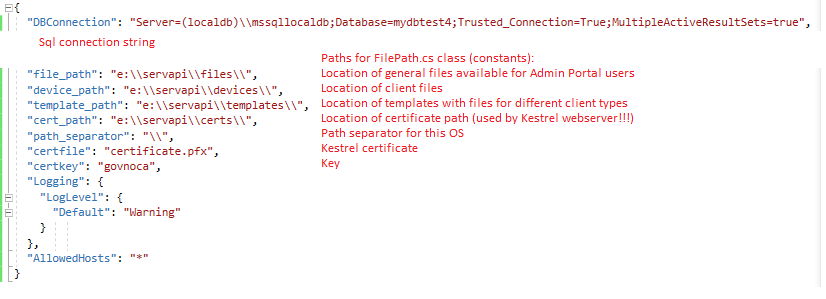
##### DBData.cs classes structure and methods



##### Router.cs methods



##### appsettings.json



**New connection string is MySQL!**

## Installation in Linux

1. Install dotnet core 3.1
2. Install mysql

Installation:

sudo apt-get update

sudo apt-get install mysql-server

Run setup:

sudo mysql\_secure\_installation utility

Start service:

sudo systemctl start mysql

sudo systemctl enable mysql

Log in:

sudo mysql -u root –p

Add dbadmin user:

CREATE USER 'dbadmin'@'localhost' IDENTIFIED BY '<password>';

GRANT ALL PRIVILEGES ON \* . \* TO 'newuser'@'localhost';

Username/Password will be used in SQL connection string.

1. Compile SERVAPI and AdminPortal
   1. Copy project folders on the Linux Server.
   2. Create folder /servapi and copy files, cert, templates, devices directories there. Assign read-write rights for current user (the one who will run webservers).
   3. Update settings in appsettings.json and appsettings.Development.json
   4. Compile (need to be in local project folder):

dotnet restore

dotnet publish -c release -o publish -r linux-x64 --self-contained true

* 1. Run AdminPortal for the first time to create database basa1 and tables:

cd publish

./AdminPortal

* 1. If you need Debug you can do “dotnet run” in the project folder.

1. Create administrator and normal user in mysql:
   1. mysql> use basa1;
   2. mysql> show tables; -> show tables to make sure they were created.
   3. Insert users:

INSERT INTO Users (Id, Username, Passwordhash, Email, Customer\_id, Fullname, Mustchangepassword, Lastpasswordchange, Userclass) VALUES (1, 'admin', '$MYHASH$V1$10000$5zKP8JYZnE6CQaI6do6w0z0pRQvgqRh+j0onojG73XRX8M9v', 'mikmon@yandex.ru', -100, 'Mikhail Monko', 0, '2020-05-23 14:38:11', 'admin');

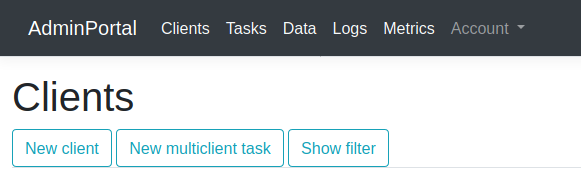
INSERT INTO Users (Id, Username, Passwordhash, Email, Customer\_id, Fullname, Mustchangepassword, Lastpasswordchange, Userclass) VALUES (2, 'mikmon', '$MYHASH$V1$10000$wUoAQXX06mZ2jvDJXTR1giQ76Fx1vkHdS3vVcqQ5+PFGxcLe', 'mikmonson@mail.ru', 1, 'Mikhail Monko2', 0, '2020-05-23 21:57:02', 'user');

1. Preparing test client

You may need to update your hosts file for prod.edgeservapi.com. The client will perform certification verification and CN in server certificate should match domain name!

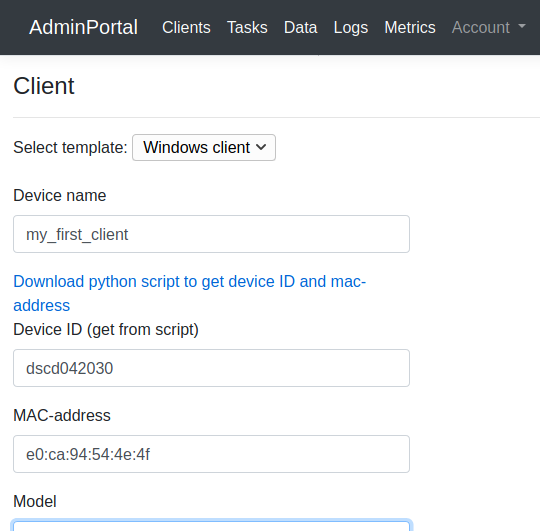
You need to create client in the DB from Admin Portal.

1. Go to Clients -> New client

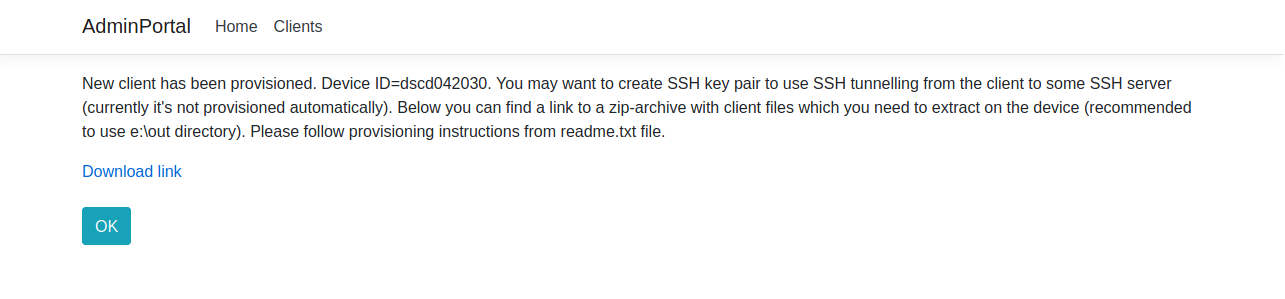


1. Fill in information about new client (all fields are mandatory).

You may need to download python script to run it on the new client to find out its identifier (figure out from S/N or MAC-address)



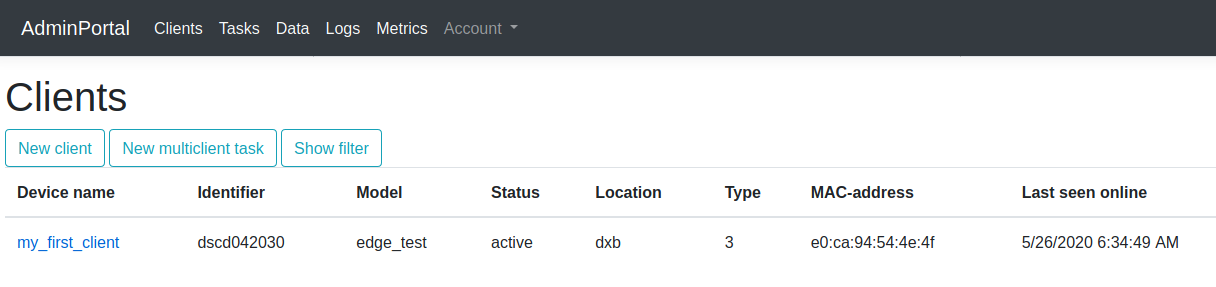
1. When new client is created you get the following message. You provided link to download zip archive with client files. The archive contains readme.txt file with short instructions on provisioning.



1. Then run the client (edgeclient.exe) from command line and look for errors.

It will send hello message to the server (name is defined in settings.txt). Make sure your hosts file is updated.

After successful hello the client’s status will change from “provisioned” to “action” in Admin Portal.



# Admin Portal code structure

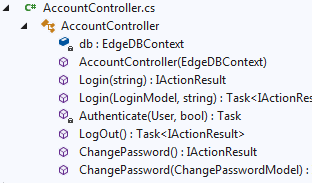
AdminPortal project is written in asp net core 3.1 and contains the following classes:

### Controllers



The project has 2 controllers.

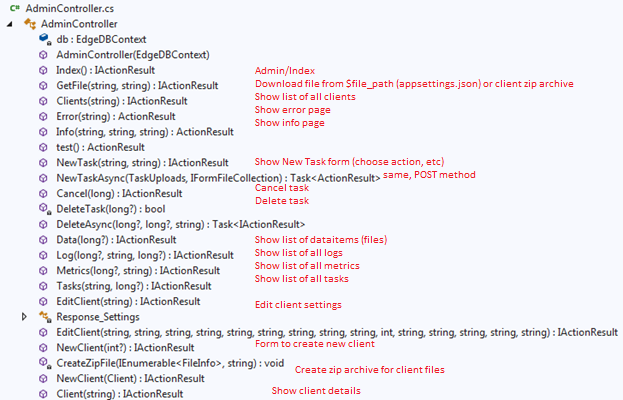
AccountController is responsible for authentication and authorization and contrains methods Login (GET/POST -> linked with Login View), Authenticate, LogOut, ChangePassword (GET/POST -> linked with View)



If a user is not authenticated or authorized he is redirected to login page.

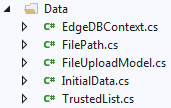
Another controller is AdminController. It’s responsible for processing requests to all other pages of the portal (/Admin/…).

Methods of AdminController:



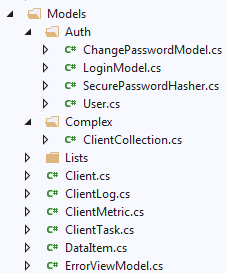
### Data classes

Data classes are: EdgeDBContext (class to operate with DB), InitialData (class to seed DB in the beginning – not in use), TrustedList (class to load/check/save trusted thumbprints from clients), FileUploadModel (class used for uploading files into task), FilePath (class with system paths).



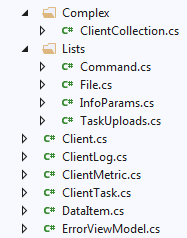
### Models

Here are the models that are used in the project:



Auth group -> contains classes related to AccountController.

Other classes are used by AdminController.



Classes Client, ClientLog, ClientMetric, ClientTask, DataItem define DB table models. They have some attributes that don’t exist in the DB and are used locally between AdminController and Views to pass some information.

ClientCollection class is a set of DB classes (Client, ClientLogs, ClientTasks,…) that describe one client -> used by Client Method/View.

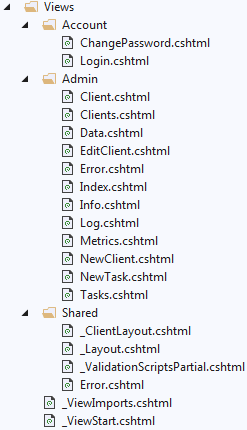
Command and File classes contain subclasses that represent lists of JSON\_file and JSON\_command (lists of strings containing filenames/types/etc or commands. They are required to convert strings from DB to proper types using JSON deserializer.

### Views

Account views -> used by AccountController.

Admin views -> used by AdminController.

Shared -> \_ClientLayout is the main master view that is used by all pages (except auth).



### Main program files



Appsettings.json contains definition of paths and certificates.

Program.cs -> main program (Kestrel webserver settings, DB init, etc).

Startup.cs -> configuration of main services (DB, authentication, MVC, etc.)

# Compiling client to different platforms

### Compiling to Windows (x64)

1. Install python 3.8 (64 bit), download and install pip, install cython and other required packages (try to run edgeclient.py to see missing libraries).
2. Install MinGW with gcc (download it as part of CodeBlocks).
3. Convert edgeclient.py to C using cython:

cython edgeclient.py –embed

1. Open edgeclient.c with Notepad++, search for wmain and change it to main (remove “w”).
2. Try to compile the file using x64 gcc from MinGW package:

x86\_64-w64-mingw32-gcc-8.1.0 -mconsole -DSIZEOF\_VOID\_P=8 -DMS\_WIN64 C:\Users\komandor\AppData\Local\Programs\Python\Python38\Scripts\edgeclient.c -IC:\Users\komandor\AppData\Local\Programs\Python\Python38\include -LC:\Users\komandor\AppData\Local\Programs\Python\Python38\libs -lpython38 -o output

1. Add python38.dll file to project directory. Include certificates and settings.txt.

### Compiling to Linux (x64)

1. Install python3 (sudo apt-get install python3)
2. Install python3-pip
3. Install cython and other missing packages (python3-pip)
4. Convert Python file edgeclient.py to C:

cython edgeclient.py –embed

1. Compile C file with gcc:

gcc -Os -I /usr/include/python3.8 -o test test.c -lpython3.8 -lpthread -lm -lutil -ldl

\*where python3.8 your python version and directory