TachoCardAuthorizer

application that enables using smart card reader for communication with TMR device

Technical documentation

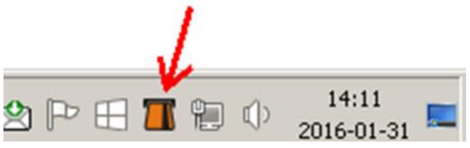
Rev. July 8th, 2016

1 Introduction

1.1 Basic information

Application enables remote authorization of Requests for reading files from tachograph by using company's tachograph memory reader device. It uses the Service on intermediate server which should provide API described further in this document. During authorization process a smart card (called company card) is used to authenticate APDU packets downloaded from the tachograph through the intermediate server.

After the application is started, it places its icon in the system tray. This icon is used for showing notifications to user. In addition, by clicking on it with right mouse button, the user can access application's context menu which provides



management options.

The application requires Java Runtime Environment (version 1.7.0 or above) to be installed on the user's machine (download is available free of charge from website https://www.**java**.com/en/**download**/).

1.2 General remarks

The application communicates with the server by using HTTP/HTTPS protocol for data transfer and JSON as default data format (unless there is said otherwise in a particular request's description). This documentation specifies which fields are required in response objects. The server may extend them with additional fields, which will be ignored by the application.

URL requests parameters (in angle brackets) are specific values related to certain requests types:

- *requestId* – authorized Request's identifier (data type: integer)
- *gpsId* – identifier of device for which the authorized Request is specified (data type: string)
- instanceId – identifier of application instance registered on server (data type: integer)
- *applicationVersion* – version of TachoCardAuthorizer application (data type: string in format *major.minor.release*, i.e. 1.2.2)

Parameters in requests descriptions are always POST parameters. GET parameters (if required) are explicitly included in URL addresses.

1.3 Application states

The application is always in exactly one of the states listed below. Every state is signalised by specific tray icon's color.

- Not logged in – the application is not logged in to the Service and awaits user's action (state signalised by red application's tray icon).
- Card reader not found – the application didn't detect any connected card reader on the user's machine and awaits user's action (state signalised by orange application's tray icon).
- Company card not found – the application didn't detect a company card in any of the card readers connected to the user's machine and awaits user's action (state signalised by yellow application's tray icon).
- Standing by – the application didn't find any Request that is to be authorized. It is, however, ready to begin authorization automatically the moment next Request is queued in the Service (state signalised by dark-green application's tray icon).
- Authorization in progress – the application is currently authorizing a Request. Forcing it to exit will cause Request to be cancelled, while disconnecting card reader from user's machine / removing company card from the connected card reader will cause Request authorization to be paused until card reader and / or company card are restored (state signalised by light-green application's tray icon).
- Connection error – the application encountered a connection error which temporarely disables data exchange with the server (state signalised by red application's tray icon).

1.4 Configuration file

After the application is started, it checkes for the *config.dat* file in its directory and either loads configuration from it, or (in case there's no such file found) creates this file and saves a default configuration (which will also be used in current application session) into it. This file's format is very similiar to INI configuration files, since consecutive lines contain settings in format *key=value* (keys are case sensitive, while values aren't). The file contains following data:

- name of system (Service running on server that intermediates in Requests authorization) that is being shown i.e. in notifications and login window's title (key in file: *systemName*)

- server's IP address / host name (key in file: *hostName*)

- port number on which server is listening for application's requests (key in file: *portNumber*)

- debug mode status – if set to *debugOn* it is enabled, while any other value causes it to be disabled; enabling debug mode forces application to skip checking if smart card in card reader is a valid company card (key in file: *debugMode*)

- secure connection mode status – specifies if application should use SSL for communication with server (enforces HTTPS) or not (enforces HTTP); if set to *true*, the application will use the former protocol, while any other value will cause it to use the latter protocol (key in file: *useSSL*)

- sleep mode status – specifies if program should halt all its communication with server and acquire status “sleeping” after longer period of time (60 minutes) without any activity (until it gets woken up by user by i.e. clicking application's icon on system tray); if set to *true*, this feature becomes enabled, while any other value will disable it (key in file: *enableSleepMode*)

- language version – Polish (for value *PL*) or english (for value *EN*); the default application's language version (set in case of missing or invalid value for this setting) is English (key in file: *languageVersion*)

- user name (stored by application) – if present, the application won't show login window and, instead, will try to login to Service using user name and password saved in the configuration file; the login window will be shown, hovewer, if user credentials saved in this file are incomplete or invalid (key in file: *username*)

- password (stored and encrypted by application) – like the user name it's not required to be stored in the configuration file (key in file: *password*). Caution! The password can't be set by editing configuration file, since it mustn't be stored as unencrypted text!

Login credentials entered by user in login window are saved in the configuration file by the application if user checks *Remember login data* option.

All changes in the configuration file should be made when the application is not running. Otherwise they may be overwritten and lost.

1.5 Request statuses

The Request may at the time have one of the statuses listed below:

- *queued*
- *authorization*
- *transfering*
- *ready*
- *cancelled*
- *error*

The application will begin authorization of first Request with the *queued* status in the queue. The Request's status will be then set by application to *authorization* and the process will begin. When the Request status changes to *transfering* or *ready*, the application will show message about successful Request authorization. If Request changes its status to *cancelled* or *error*, the authorization process will be stopped and application will show proper message (depending on the Request status). Aside from these cases, the application doesn't react to any Request statuses (even if they are not listed above).

The application checks tachograph access Request status by using HTTP requests described in sections 8, 9 and 10.

2 HTTP requests descriptions

This section describes HTTP requests that application sends to server in order to authorize Requests for downloading files from tachograph. These HTTP queries include authorization of access to tachograph files and user authentication (required by Service provided by server for Requests management), as well as downloading application metadata (i.e. its new version availability). The server must handle HTTP requests used by the application as described in this section so the application would be able to use them properly. Server's HTTP responses must include data specified in response's descriptions for each HTTP request, but any additional data (that is not mentioned in these descriptions) will be simply ignored.

The application periodically checks for queued Requests or APDU packets on server. It is assumed that the server manages queue of Requests such that every Request can have one of the statues described in section 4 at the time. A single instance of application and company card may authorize one and only one Request from that queue at once.

To handle Requests, server receives data packets (APDU) from TMR through GPRS terminal and provides them to application, which periodically checkes for the APDU packets related to the Request it currently authorizes. Downloaded APDU packets are transmitted by the application to company card, which's response is then returned to the server instantly. Authorization process ends when Request's status gets changed by server (which means the Request was either cancelled or its authorization was completed).

In order to avoid conflicts resulting from simultaneous attempts to authorize the same Request by two or more application instances (which are logged into the Service using identical credentials), the application uses instance registration, which allows only one application instance per user account to authorize Requests at the given time. This mechanism, however, isn't mandatory, so if HTTP requests descibed in 6, 7, 8 and 8 are not implemented on server, the application will not raise any errors and will work just as fine (only that there is a risk of conflicts that were mentioned above to arise).

Server's HTTP responses should always have status 200 OK, unless the server intends to signal an error. If then HTTP response with the code of 300 or greater is returned to application (it doesn't support redirection status codes), it will show an error notification informing about the status code returned by server in its response. In case of receiving reponse with status code 403 Forbidden, the application will show more specific message about lack of permission needed to perform a request.

2.1 Login

Method: POST

URL: /account/zaloguj/

Parameters: username, password

Response description: A JSON object containing information about login operation status.

Response data structure:

```
class LogInMessage
{
    string message;
}
```

Comments: Before fetching data about Requests that are to be authorized, application tries to log in as user to Service in order to gain access to server's authorization API. If application is running in debug mode, it is possible to bypass this action by checking appropriate checkbox in the log in window. While credentials aren't sent in JSON format as HTTP request's data, the server's response data is. If JSON response's object's *message* field contains value *ok* (case insensitive), it is acknlowledged as successful login attempt, and from that moment application will be using cookies from this server's response in all consecutive HTTP requests. On the other hand, any *message* field's value different from that mentioned above will be interpreted as failed login attempt, which will be signalised by the application with the proper notification.

If the *message* field contains followin string: *Your limit of tries is achieved, try again later*, the application will show more specific notification about achieving maximum login tries by user.

Response data example: {"message": "ok"}

2.2 Logout

Method: GET

URL: /account/wyloguj/

Parameters: none

Comments: Terminating the application causes it to log out from the Service, where it was logged in as the user. However, failed logout attempt doesn't cause any error, and since there is no response check for this HTTP request, it is not required for application to work properly.

2.3 Fetching registered application instances list

Method: GET

URL: /api/tachograph/v1/client-status

Parameters: none

Response description: a JSON object containing list of objects describing registered instances, as well as their total count.

Response data structure:

```
class RegisteredClientsDataMessage
{
    RegisteredClientData[] items;
    int total_count;
}
class RegisteredClientData
{
    int id;
    string name;
```

```
        string version;
        string username;
        string updated_at;
        boolean connected;
        boolean reader_connected;
        boolean card_inserted;
    }
```

Comments: Although response for this request contains list of objects, the application will not actually register itself on server if there is already another instance registered. Because of that this list will always either be empty or contain exactly one element.

If this request is not implemented on the server (which should then return response with status code 404 Not Found), the application will not be using instance registration mechanism and consequently neither this nor any request described in 7, 8 and 8 will be used during its lifespan.

The instance registration mechanism serves only for synchronization purpose (disables any „races” to enqueued Requests between application instances) and is not connected with user registration or authentication.

Response data example:

{"items": [{"card_inserted": true, "connected": true, "id": 7, "name": "TachoCardAuthorizer", "reader_connected": true, "updated_at": "2016-03-17T18:18:20.361509+00:00", "username": "test", "version": "1.2.3"}], "totalCount": 1}

2.4 Application instance registration

Method: POST

URL: /api/tachograph/v1/client-status

Parameters: a JSON object of the following structure:

```
    class ApplicationRegistrationDataMessage
    {
        string name;
        string version;
        boolean busy;
        boolean card_inserted;
        boolean reader_connected;
    }
```

Response description: a JSON object of structure specified as `RegisteredClientData` (described in 2.3).

Comments: Server's response should include data sent by application in the request (fields *name*, *version*, *reader_connected* and *card_inserted* from request's data object are corresponding to similiar fields form response's data object).

If application will not be able to register its instance on the server while appropriate API is available, it could not perform any authorization operations on Requests. On the other hand, if HTTP request described in 6 is not implemented on server, this one will never be used and session registration will not be required to perform authorizations.

The *busy* field indicates if application is currently authorizing a Request.

Response data example:

{"card_inserted": true, "connected": true, "id": 7, "name": "TachoCardAuthorizer", "reader_connected": true, "updated_at": "2016-03-17T18:18:20.361509+00:00", "username": "test", "version": "1.2.3"}

2.5 Application instance status actualization

Method: POST

URL: /api/tachograph/v1/client-status/<instanceId>

Parameters: same as for 7.

Response description: same as for 7.

Comments: If HTTP request described in 6 is implemented on server, application will be updating its status on server periodically by using application instance status actualization request. It will also occur upon (dis)connecting card reader, inserting card into reader of removing card from reader. An *instanceId* parameter can be retrieved from field *id* in reponse object for HTTP request described in 7 (and 6 if current application instance is registered on the server at the time).

Response data example:

{"card_inserted": true, "connected": true, "id": 7, "name": "TachoCardAuthorizer", "reader_connected": true, "updated_at": "2016-03-17T18:18:20.361509+00:00", "username": "test", "version": "1.2.3"}

2.6 Application instance unregistration

Method: DELETE

URL: /api/tachograph/v1/client-status/<instanceId>

Parameters: none

Comments: The application can unregister not only itself, but also other registered instances run on other machines (as long as they are actually registered on server and are bound to the same user account).

2.7 Fetching Requests queue

Method: GET

URL: /api/tachograph/v1/queue?limit=1&status=queued&online=true

Parameters: none

Response description: A JSON object containing device's GPS ID plus Request's ID and status.

Response data structure: object *QueuedOperationsMessage*

```
class QueuedOperationsMessage
{
    OperationData[] items;
}
class OperationData
{
    int id;
    OperationDetails request;
}
class OperationDetails
{
    string status;
```

```
        string status_message;
        string device;
        int progress;
    }
```

Comments: If there is at least one queued Request, *items[0].request.device* must contain an identifier of GPS device which the Request is relevant to, and *items[0].id* – this Request's identifier. If there are no queued Requests, the *items* field must be an empty list. If the Request's status is not *queued*, the application won't start authorizing it.

The application always checks information about first Request from the queue regardless of *items* list size (on condition that it is greater than zero), which is forced with the *limit* parameter in query URL. Consequently this information is limited to data about Requests that are actually queued (URL's *status* parameter) and are relevant to active devices (URL's *online* parameter).

Response data example: {"items": ["request": {"status": "queued", "device": "123456789"}, "id": 98765]}

2.8 Retrieving Request's data

Method: GET, POST

URL: /api/tachograph/v1/requests/<requestId>

Parameters: none (for GET query) or modified response for GET request of the same type (for POST query)

Response description: A JSON object that contains, inter alia, a *status* field which must be a string value *queued* when retrieved from GET request's response (see below).

Response data structure:

```
    class OperationDetails
    {
        string status;
        string status_message;
        string device;
        int progress; //Not obligatory – see comments
    }
```

The response has always the same structure for both GET and POST HTTP requests.

Comments: Starting an authorization is caused by changing Request's status, which is the purpose of this HTTP request. First a GET query is used in order to fetch Request's data (which's ID has been taken from response to request described in 8). Then JSON response object's *status* field is changed from *queued* to *authorization* and – unchanged anywhere else – sent in POST HTTP request back to the server.

The same mechanism serves for application to change Request's statuses to *cancelled* (when i.e. application was shut down by user during authorization) or *error* (if there were problems that made further authorization impossible). On such events field *status_message* in HTTP request's data object will contain information regarding cause of Request's status change.

There is additional functionality realized with this query that Service may or may not implement – operation completion percentage progress actualization. After GET query is executed, the application can modify response object's *progress* field and return it with POST request. It normally does so every time APDU packet is processed. As the *progress* field represents percents, it shall contain value from range 0-100.

Parameters example: {"status": "authorization", "device": "123456789"}

Response data example: {"status": "queued", "device": "123456789"}

2.9 Getting Request's status

Method: GET

URL: /api/tachograph/v1/apdu/<requestId>/status

Parameters: none

Response description: A JSON object containing information about awaiting APDU packets and Request's current status.

Response data structure:

```
class OperationStatus
{
    int apdu_count;
    OperationDetails request;
}
```

Comments: This HTTP request is crucial control tool for the authorization process. If *request.status* field's value (which contains tachograph access Request's status) is *authorization*, the application will repeat this HTTP request periodically until:

- *apdu_count* field will contain value greater than 0 (which causes application to send APDU packet dequeueing request to server, but always one at the time, regardless of the *apdu_count* field's value)

- *request.status* field's value is *transfering*, which application ackonwledges as successful finish of authorization process

- *request.status* field's value is different from *authorization* and *tranfering* (i.e. *cancelled* or *error*), which causes application to give up on current authorization

After a successful company card response HTTP request, the application queries server for Request's status again in order to decide on its further authorization.

Response data example:

{"request": {"status": "queued", "device": "123456789"}, "apdu_count": 0}

2.10 APDU packet dequeue

Method: POST

URL: /api/tachograph/v1/apdu/<gpsId>/dequeue

Parameters: none

Response description: A JSON object containing another JSON object, which consists of APDU packet's body and number.

Response data structure:

```
class DequeuedAPDUMessage
{
    APDU packet;
}
class APDU
{
    int seqnum;
    string body;
}
```

Comments: If server returns an invalid APDU packet, it will cause application to show error notification and change authorized Request's status to *error*, which will lead to terminating further authorization of that Request.

Response data example: {"packet": {"body": "123456789012", "seqnum": 1}}

2.11 Commiting APDU packet

Method: POST

URL: /api/tachograph/v1/apdu/<gpsId>/commit

Parameters: none

Response description: A JSON object that contains status of commiting APDU packet query's realisation.

Response data structure:

```
class RequestStatusMessage
{
    string status;
}
```

Comments: The aplication sends this query after dequeuing APDU packet and before sending company card's response for that packet. If response's object *status* field's value is different than *OK*, it will cause an internal application error and prevent from sending card's response for the last downloaded APDU packet. Instead, the application will send Request status query and carry on authorization process.

Response data example: {"status": "OK"}

2.12 Sending company card's response for APDU packet

Method: POST

URL: /api/tachograph/v1/apdu/<gpsId>

Parameters: A JSON object that contains company card's response for APDU packet and this packet's number:

```
class CardResponseMessage
{
    string packet;
    int seqnum;
}
```

Response description: Same as response to APDU packet commit request. In case of *status* field's value being different from *OK,* an error is raised in application, but authorization is continued anyway.

Comments: Packet's number in the request's data object is always equal to number of APDU packet downloaded by application in current APDU fetching cycle.

Parameters example: {"packet": "123456789ABCDEF0", "seqnum": 1}

Response data example: {"status": "OK"}

2.13 Retrieving application metadata

Method: GET

URL: /api/tachograph/v1/tachoappinfo?current_version=<applicationVersion>

Parameters: none

Response description: A JSON object. According to current specification it contains only one field – string with the latest application's version.

Response data structure:

```
class ApplicationMetadataMessage
{
    string latest_app_version;
}
```

Comments: The application retrieves its latest version number and compares it with its running instance's version number. If they are different, it shows a notification about new version's availability. As the application doesn't actually check if the version from metadata request is newer than its own, any version number that differs from application's current version will cause it to show that notification.

The application instance send its version as GET parameter (*applicationVersion* parameter).

This request is not required for the application to work properly.

Response data example: {"latest_app_version": "1.0.2"}