Roadside interface to AutoPASS HUB

April 27th 2021

Introduction

The Roadside interface to AutoPASS HUB is basically the same as for CS Norway. File formats can be found in:

AutoPASS 4.3 Data formats overview and AutoPASS 4.3 formats Appendixes

The following changes have been introduced with AutoPASS Core:

1. File transfer protocol has been changed from FTP to SFTP
2. A new reject file has been introduced.

SFTP interface

SFTP accounts

- There is one SFTP account (and home directory) per TC and per Roadside Provider
- Username convention: <Roadside Provider>-<TC actor ID>. Example: QFree-100030
- The Roadside Provider shall authenticate using SSH keys (no password). The same key pair shall be used for all TCs that the Roadside Provider handles
    - The public key shall be sent to the AutoPASS SPOC

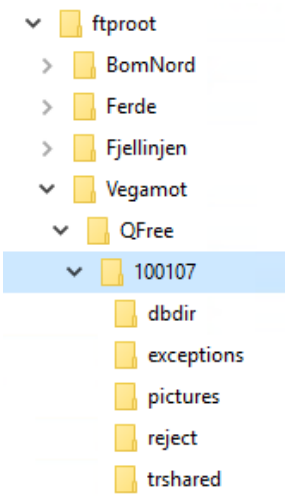
Directory structure

Roadside components (concentrators) upload files to and download files from the following sub-directories. This is the same directory structure as for CSNorway:

- Directories to which roadside uploads files:
    - trshared: contains transaction files (tr)
    - pictures: contains image files
    - exceptions: contains exception files

- Directories from which roadside downloads/views files:
    - dbdir: contains for example status lists, issuer lists etc.
    - reject: files uploaded from roadside that have been rejected

Sample



Processing of uploaded files

Upload complete.

The Roadside Adaptor will start processing a file as soon as the write handle has been released by the SFTP client. This means that the client should not upload temporary files and rename (or move) them when the upload has completed. The Roadside Adaptor might try to handle even temporary files. An exception to this has been implemented: files with a filename ending with .tmp will not be handled by the Roadside Adaptor. This means that the client mayupload files temporarily as .tmp files and then rename them - but again: it is recommended to upload files with their final filename.

The Roadside Adaptor deletes the file when it has completed the file processing.

Rejected files.

Rejected files will be moved to the "reject" folder. A new feature has been implemented for the AutoPASS Core solution where a specific file will be madeavailable together with the original file - a file that describes the detailed reason(s) why the file was rejected.

A roadside file can be rejected by AutoPASS HUB or by AutoPASS IP according to attached “Interface specification for AutoPASS HUB 1.1#Reject”. This will producea json file with reject reasons.

Filename convention

The reject file shall have the same filename as the original (rejected) file - but with the file extension .json added to the original filename.Samples:

- Original transaction file:

  tr100007_202005251210408_09.str reject file tr100007_202005251210408_09.str.json

- Original picture file:

  Pic100007_P001_L01_F1_XEX_D2020_04_14_T11_59_38_N00000021.jpg reject file **Pic100007_P001_L01_F1_XEX_D2020_04_14_T11_59_38_N00000021.jpg.json**

File format

The json file format and sample json files are described in “Interface specification for AutoPASS HUB 1.1#Reject”

Rejected files from Roadside

The client can use this endpoint to reject a downloaded file which is not valid. The original file and the error information will be returned to the original sender.

reject

The /reject endpoint currently only supports roadside files, that is tr files and picture (jpg) files.

Endpoint description

Method: POST

Input parameters:

Parameter	Mandatory	Description	Parameter Type	Data type
transactionId	yes	Refers to the file. The transactionId can be found in the AMQP message or the file URI	query	string
Authorization	yes	Authorization token	header	string
rejectMessage	yes	A structured text element containing the detailed error description. See format requirement below. Content type: application/json	body	string

Response contents: empty

JSON format requirement

The JSON string shall contain the following global parameters:

Parameter name	Description	Mandatory value	Format	Sample
transactionId	Reference to the original file (same as the query parameter)	yes	string (UUID)	"51b68aa0-b096-417c-b074-0f2d4b7e83e2"
filename	Filename	yes	string (Filename)	"tr100007_201910251210065_09.str" or "Pic100007_P001_L01_R1_XEX_D2019_09_04_T11_59_38_N00000021.jpg"

originalSender	Original sender of the file	yes	string (AutoPASS Actor ID)	"100007"
downloadTime	Timestamp for when the file was downloaded	yes	yyyy-MM-dd'T'HH:mm:ss (ISO 8601)	"2019-11-01T13:00:34"

Validation rules:

- originalSender must be equal to Actor ID in filename.

The JSON string shall also contain an array of error elements (one or more) - where each error shall refer to a specific line/record in the original file. The error element shall contain the following parameters:

Parameter name	Description	Mandatory value	Format	Sample
lineNumber	A line is a string in the text file that ends with a line separator. First line gets line number 1. This parameter will have no value in the case of a jpg file.	no	integer	23
chargingPoint	Number of the charging point where the passage has taken place	no *)	string (3 digits)	"065"
lane	Number of lane where passage took place	no *)	string (2 digits)	"02"
passageTime	Time when passage took place	no *)	yyyy-MM-dd'T'HH:mm:ss (ISO 8601)	"2019-11-01T13:00:34"
OBU	OBU id	no *)	Refer to "4.3 AutoPASS Data Formats.pdf, appendix A8"	"97800300031200123A"
errorAtPosition	Start position for detected error. First character on a line has position 1. A value 'null' means that the error is not related to a specific position - it applies to the entire line (or file).	no	integer	130
errorCode	See table below	yes	integer in the range 100 to 999	178
errorDescription	Textual description of the error detected for the actual line	yes	string	"An error description"

*) The parameter shall correspond exactly to the parameter in the original tr file or picture file (filename). Refer to 4.3 AutoPASS Data Formats.pdf, appendix A8 and A9

Note on mandatory

All parameters shall be present in the JSON string. Missing values shall be represented as 'null' (without quotes).

Error codes (the list is not exhaustive and is subject to changes):

Error code	Error name	Description
General		
101	Duplicate	The same file has been processed already
Tr file		
201	Invalid file length	
202	Corrupt data	Use for example if the line is corrupted or not in line with the specification
203	Missing parameter value	
204	Invalid parameter value	Use also in case, for example the operator code, toll plaza or lane is unknown
205	Invalid parameter format	Use for example if the nationality does not comply with ISO 3166 Alpha 2
206	Match failure	Parameter does not match the value of the parameter in the filename
Picture file		
301	Unreadable picture	
302	Corrupt picture file	
303	Missing picture file	
304	Invalid parameter value	Filename parameter (for example charging point or lane number) is invalid
305	Match failure	Picture imposed parameters does not match with the filename

Samples:

Sample JSON for tr file


```
{
        "transactionId" : "51b68aa0-b096-417c-b074-0f2d4b7e83e2",
        "filename" : "tr100007_201910251210065_09.str",
        "originalSender" : "100007",
        "downloadTime" : "2019-11-01T13:00:34",
        "error" :
        [
            {
            "lineNumber" : 23,
                "chargingPoint" : "065",
                "lane" : "02",
                        "passageTime" : "2019-11-01T13:00:34",
                        "OBU" : "97800300031200123A",
                        "errorAtPosition" : 44,
                        "errorCode" : 204,
                        "errorDescription" : "The ServiceNumber is not numeric ("Ø~$")"
                },
                {
            "lineNumber" : 39,
                "chargingPoint" : "066",
                "lane" : "01",
                        "passageTime" : null,
                        "OBU" : "97800300031200124A",
                        "errorAtPosition" : 8,
                        "errorCode" : 203,
                        "errorDescription" : "Missing passage time"
            },
                {
            "lineNumber" : 99,
                "chargingPoint" : null,
                "lane" : null,
                        "passageTime" : null,
                        "OBU" : null,
                        "errorAtPosition" : null,
                        "errorCode" : 201,
                        "errorDescription" : "Line length (1024) is not correct (512)"
            }
    ]
}
```


Sample JSON for picture file


```
{
        "transactionId" : "51b68aa0-b096-417c-b074-0f2d4b7e83e2",
        "filename" : "Pic100007_P001_L01_R1_XEX_D2019_09_04_T11_59_38_N00000021.jpg",
        "originalSender" : "100007",
        "downloadTime" : "2019-11-01T13:00:34",
        "error" :
        [
            {
            "lineNumber" : null,
                "chargingPoint" : "001",
                "lane" : "01",
                        "passageTime" : "20190904115938000",
                        "OBU" : null,
                        "errorAtPosition" : null,
                        "errorCode" : 301,
                        "errorDescription" : "Unable to read picture - snow storm"
                }
    ]
}
```

Interface specification for AutoPASS HUB 1.1

Introduction

AutoPASS HUB offers two application programming interfaces:

- AMQP interface for exchange of meta data
- File REST API for transfer of files (data)

The AMQP interface has been implemented with RabbitMQ (www.rabbitmq.com).

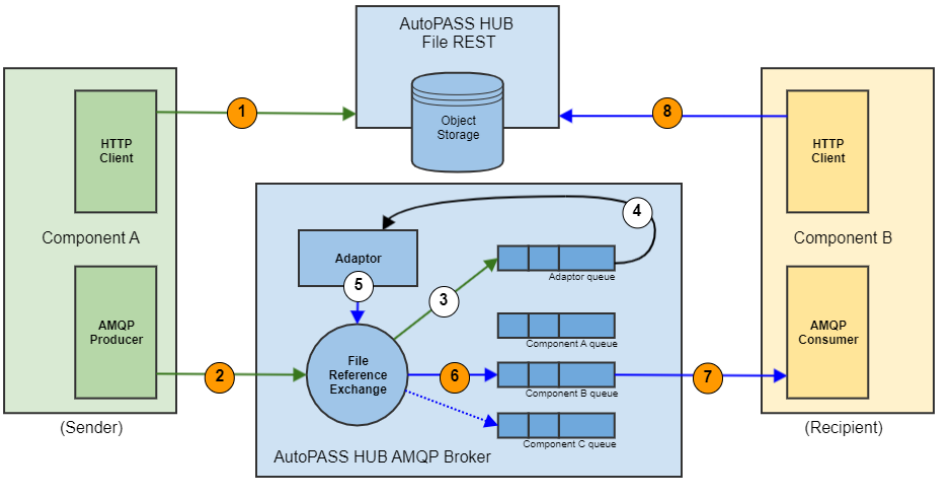
Definitions

Expression	Description
AutoPASS component	An AutoPASS components integrates with the AutoPASS HUB and is one of the following: • AutoPASS IP • TC • TSP • ANPR/MIR (does not utilize the AutoPASS HUB File REST API, but uses the ImageDB File REST API instead)

Version log

Version	Date	Change description	Change order
1.0	17.10.2019	Initial version (moved from AutoPASS HUB document space)	
1.1	05.11.2019	New endpoint: reject	HUB SSA-V COO#20

Overview



The figure illustrates how the AutoPASS components integrate with the AutoPASS HUB and shows a typical scenario where Component A sends data (an AutoPASS file) to Component B. The steps are as follows:

1. The sender, Component A (for example AutoPASS IP) is about to send a file (for example a TIF file) via AutoPASS HUB and starts this process by uploading the file via the AutoPASS HUB File REST API. The File REST API returns a URI that can be used to download the file.
2. The sender does not need to resolve the recipient and simply creates an AMQP message containing the download URI and publishes this message to the File Reference Exchange.
3. The Adaptor (for example the AutoPASS IP Adaptor) subscribes to all messages from the associated component, and the AMQP Broker delivers the message to the Adaptor's queue.
4. The Adaptor receives the AMQP message, downloads and validates the file (details not shown here).
5. The Adaptor publishes a new AMQP message containing the same URI to the File Reference Exchange. The Adaptor has now resolved the recipient (typically based on the filename) and uses a routing key that identifies the recipient(s).
6. The File Reference Exchange delivers the message to Component B's queue (a message might also be delivered to more than one queue).
7. The recipient (for example a TC) has bound to Component B queue and receives the AMQP message containing the download URI.
8. The recipient downloads the file from the File REST API

AMQP properties

The AMQP exchange, queues and bindings will be managed by the AutoPASS HUB vendor. The AMQP properties can be found here.

AMQP message format

Payload

The payload of the AMQP message shall be a JSON string, formatted as follows (the sequence of the parameters is not important). The parameter values are not real (for demonstration purposes only).

AMQP message format

```
{"fileUri":"https://hubweb.autopassops.no/hub_file_rest/download?transactionid=1ccbf1cd-5c82-44be-8426-
1675819e4688&filepath=TIF100044201801010001_30C008_130001&se=2018-08-01T12:00:34&sip=0.0.0.0-255.255.255.255
&sig=jZTiEhjNSu8fts4wmNfGMDV5OpTRj/P3SRDCaZcjziM=",
"checksum":"23rZjcYTabWUrHm6ySAbYvfouxBtLQV0mxSe2VAPMc0=",
"transactionId":"1ccbf1cd-5c82-44be-8426-1675819e4688",
"fileType":"tif"}
```

The AutoPASS HUB will use the parameters as follows:

- fileUri: to download the file (for further processing) and to infer sender and recipient (based on filename). Note that the fileUri has to be exactly the same as the client got as response from the upload request. The URI will be validated by the download endpoint.
- checksum: to verify file integrity.
- transactionId: reference to the file. The transactionId will also be used to track all file events.
- fileType: AutoPASS file type

Checksum

The checksum parameter of the AMQP message shall be generated as a SHA256 digest of the file contents. Following is a java implementation of the checksum generation.

CheckSum.java

```
package no.autopassops.tools;

import java.io.*;
import java.security.MessageDigest;
import java.util.Base64;

public class Checksum {

    private String checksum;

    public String getChecksum(InputStream in) throws Exception {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        try {
            byte[] buf = new byte[4096];

            for (;;) {
                int count = in.read(buf);
                if (count < 0) {
                    break;
                }

                digest.update(buf, 0, count);
            }
        } finally {
            if (in != null) {
                try {
                    in.close();
                } catch (Exception e) {
                }
            };
        }
        byte[] sha256 = digest.digest();
        byte[] base64 = Base64.getEncoder().encode(sha256);

        checksum = new String(base64);
        return checksum;
    }

    public static void main(String[] args) {
        try {
            FileInputStream fi = new FileInputStream("C:\\Project\\AutoPASS
HUB\\Testdata\\TIF100007201909080040_100007_130001");
            Checksum cs = new Checksum();
            String checksum = cs.getChecksum(fi);
            System.out.println("Checksum: " + checksum);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Routing key

The routing key shall be formatted as follows:

`<source>.<id>.<datatype>.<target>`

Where:

- source is the actual AutoPASS component that is producing the message. This is one of the following:
  - acfc
  - easygo
  - roadside
  - ip

    - tc
    - tsp
- id is the unique ID for the source (component). Note that there might be more than one instance of the same component.
- datatype is the AutoPASS file type and is one of the following:
    - tr
    - tif
    - tic
    - hgv
    - hgc
    - nat
    - nac
    - nbs
    - act
    - ait
    - tst
    - alc
    - img
    - ex
    - alm
    - obustatusfile
    - tariffile
    - videotextfile
- All AutoPASS components (except AutoPASS HUB itself) shall always set target to "hub".

The AMQP producer shall provide the routing key according to this format when it sends the AMQP message to the File Reference Exchange. For example:

- ip.301.tif.hub: AutoPASS IP is sending a TIF file
- tc.312.tic.hub: TC with Id 312 is sending a TIC file

Message subscription

Message subscriptions takes place via routing bindings applied to the message recipient's queue. Bindings are managed by the AutoPASS HUB vendor only, meaning that the recipient needs to contact the AutoPASS HUB vendor in order to subscribe to new messages. Note that the default bindings (for example for AutoPASS IP or for a TC) will cover all mandatory file types.

File REST API

General

The description of the current version of the File REST API can always be found online on https://hubtest4web.autopassops.no/hub_file_rest/swagger-ui.html. This service can also be used to manually try all endpoints.

The descriptions have been extracted and pasted into the sections below, one for each endpoint.

The following pseudo code shows how a test client typically works, using the File REST API. The login, upload and download endpoints are described below.

Pseudo code file REST API client

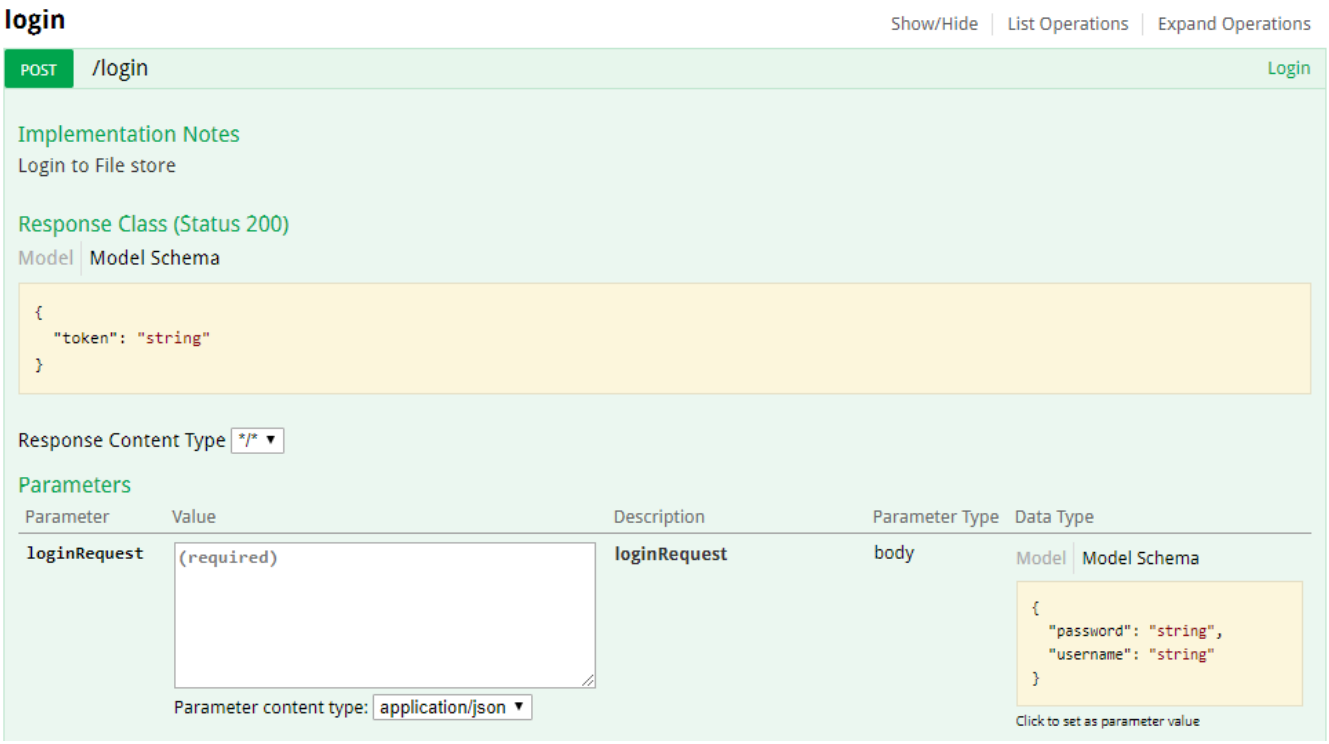
```
// Upload file
String token1 = login(username1, password1);
String transactionId = UUID.ramdomUUID().toString();
String downloadURI = upload(transactionId, new File("test.jpg"), token1);


// Download file (as another client)
String token2 = login(username2, password2);
File downloadedFile = download(downloadURI, token2);
```

Login

The client shall login with username and password to get access to the File REST API. The login request will return an authorization token that has to be included with all following upload, download or exists requests. The authorization token has an expiration time and the client will have to login again if the token has expired.

Endpoint description

The Swagger UI describes the login endpoint as follows:



HTTP status codes

The login endpoint may return the following http status codes:

Code	Name	Description	Error handling
200	OK	Login successful	
401	Unauthorized	Invalid username and/or password	Contact the SPOC

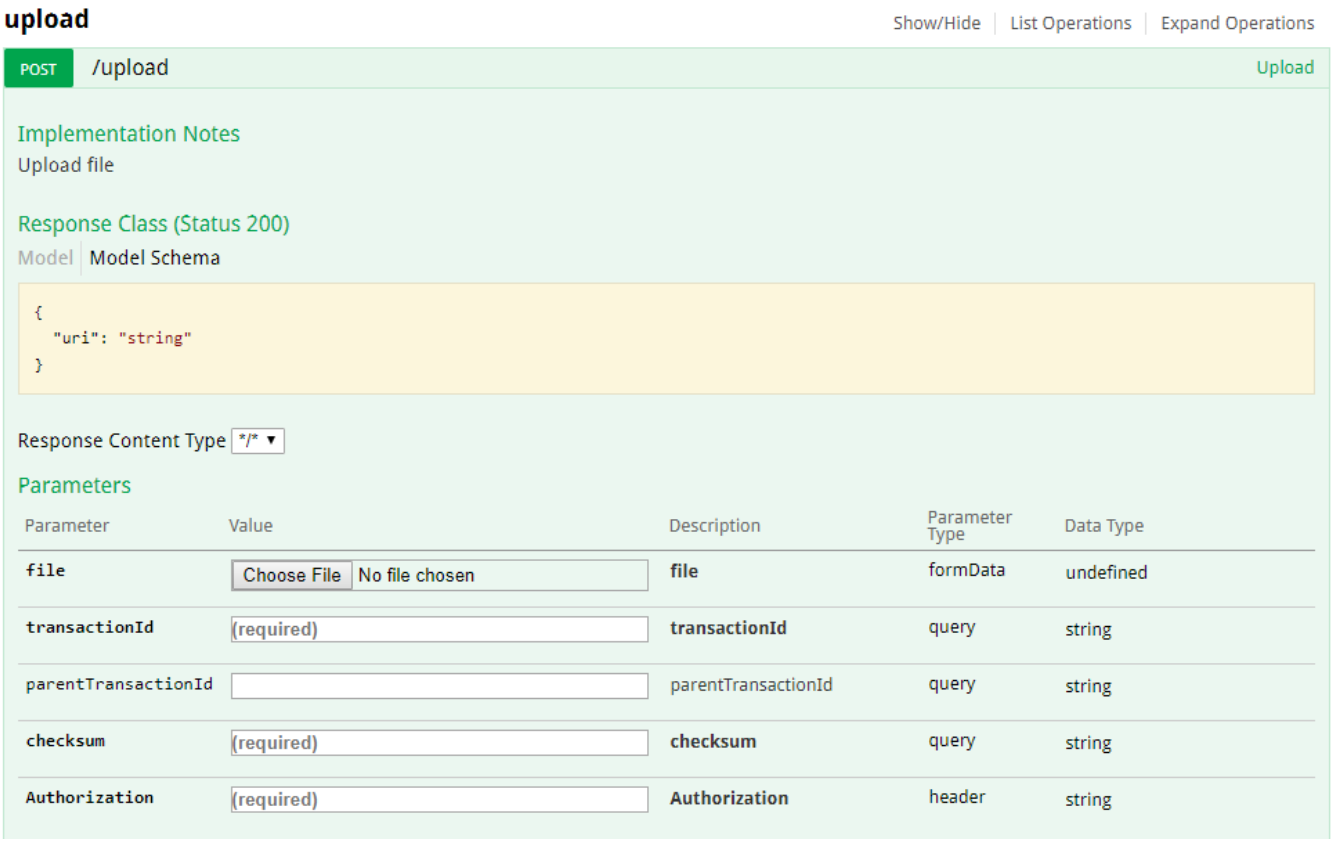
Upload file

The upload endpoint can be used to store a single file in the AutoPASS HUB filestore. The resulting URI shall be put into the AMQP message that is going to the message exchange. The download URI contains the following parameters (based on the Azure File REST principles):

- transactionId: The UUID that was generated by the client
- filepath: path and name for the file that was uploaded
- se: expiration time. The URI will not work after this time
- sip: the IP range to be used for IP screening of the client
- sig: HMAC SHA 256 signature generated for the URI parameters. The siq will be used to assure the integrity of the URI.

Endpoint description

The Swagger UI describes the upload endpoint as follows:



Authorization

The word "Bearer" and a space has to be entered before the Authorization token (returned from the login endpoint).

Java sample

The following sample (Java) code shows how to upload a file using this endpoint:

JavaUploadTest.java

```
import java.io.File;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.HttpResponse;
import org.apache.http.entity.mime.content.FileBody;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;

...

String img = "file_to_be_uploaded.jpg";

File file = new File(img);

HttpPost request = new HttpPost(resttestUrl + "upload");

MultipartEntityBuilder builder = MultipartEntityBuilder.create();

builder.setMode(HttpMultipartMode.BROWSER_COMPATIBLE);

FileBody fileBody = new FileBody(file);
builder.addPart("file", fileBody);
builder.addTextBody("transactionId", transactionId); builder.addTextBody("checksum", checksumString); builder.
addTextBody("cameraId", "0"); request.setConfig(requestConfig); request.addHeader("Authorization", "Bearer " +
token);

HttpEntity multipart = builder.build();
request.setEntity(multipart);

HttpResponse response = httpclient.execute(request);
```

HTTP status codes

The upload endpoint may return the following http status codes:

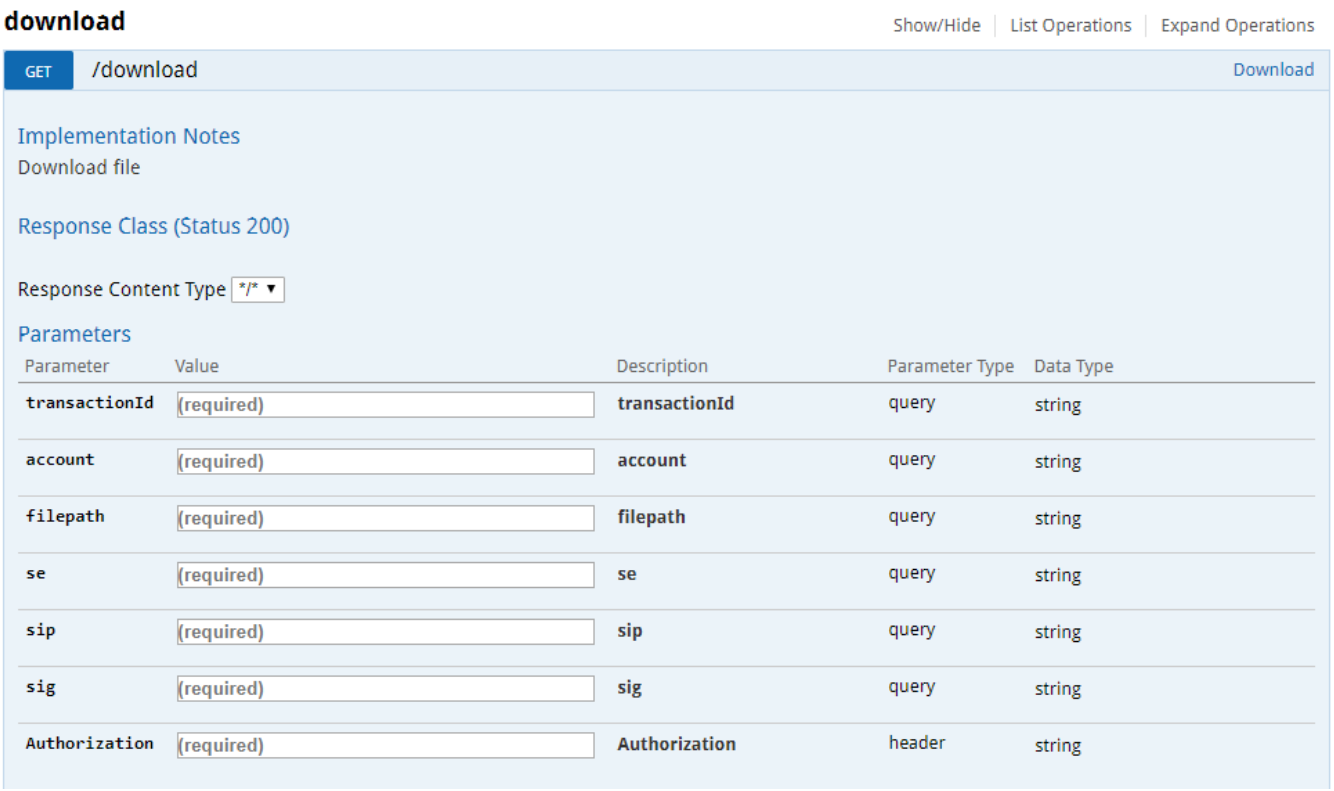
Code	Name	Description	Error handling
200	OK	Upload successful	
400	Bad request	One or more of the input parameters were invalid	Resolve error and try again
401	Unauthorized	Authorization token is not valid	Login to get a new token
409	Conflict	The file does already exist	Do not retry
422	Unprocessable entity	The file is invalid	Resolve error and try again

Download file

This endpoint can be used to download a file using a downloadURI. Note that all information is contained in the downloadURI, including the expiration time. But the client needs to provide a valid authorization token in order to access the File REST API.

Endpoint description

The Swagger UI describes the download endpoint as follows:



HTTP status codes

The download endpoint may return the following http status codes:

Code	Name	Description	Error handling
200	OK	Upload successful	
400	Bad request	One or more of the input parameters were invalid	Resolve error and try again
401	Unauthorized	Authorization token is not valid	Login to get a new token
410	Gone	The file does not exist	Do not retry

Filename

Note that the download URI contains an object reference to the file and not the original filename. The original filename can be found in the Content-Disposition response-header field. An example of this field is:

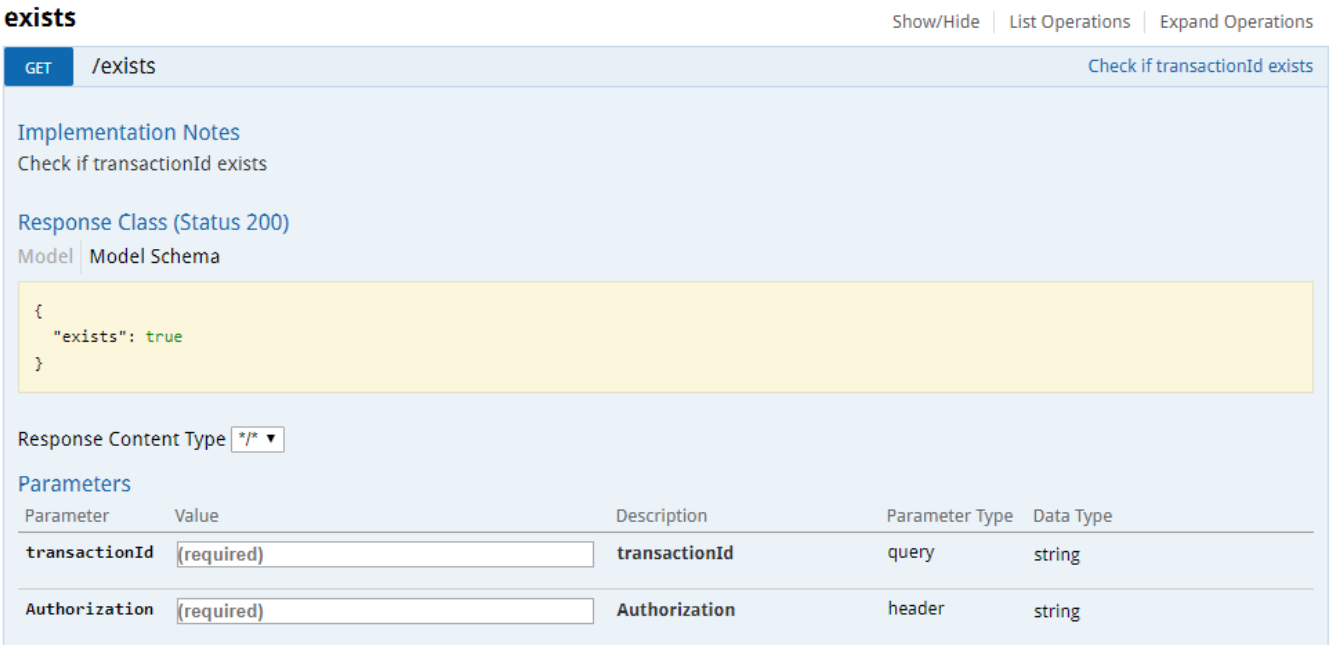
```
Content-Disposition: attachment; filename=downloaded_file.txt
```

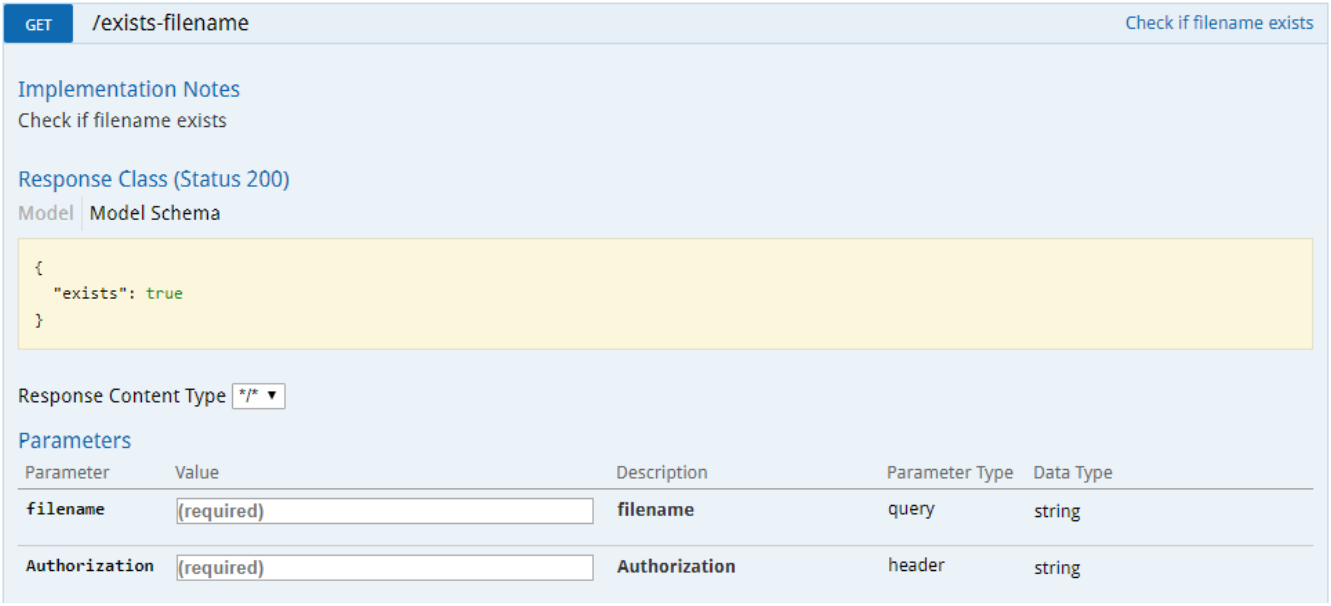
Check if file exists

The client can use this endpoint to check if a file has been successfully uploaded to the filestore. The transactionId that was submitted together with the upload attempt - or the filename can be used as input to this endpoint. Note that the transactionId is generated by the client.

Endpoint description

The Swagger UI describes the exists endpoints as follows:





HTTP status codes

The download endpoint may return the following http status codes:

Code	Name	Description	Error handling
200	OK	Upload successful	
401	Unauthorized	Authorization token is not valid	Login to get a new token

Reject

The client can use this endpoint to reject a downloaded file which is not valid. The original file and the error information will be returned to the original sender.

reject

The /reject endpoint currently only supports roadside files, that is tr files and picture (jpg) files.

Endpoint description

Method: POST

Input parameters:

Parameter	Mandatory	Description	Parameter Type	Data type
transactionId	yes	Refers to the file. The transactionId can be found in the AMQP message or the file URI	query	string
Authorization	yes	Authorization token	header	string
rejectMessage	yes	A structured text element containing the detailed error description. See format requirement below. Content type: application/json	body	string

Response contents: empty

JSON format requirement

The JSON string shall contain the following global parameters:

Parameter name	Description	Mandatory value	Format	Sample
transactionId	Reference to the original file (same as the query parameter)	yes	string (UUID)	"51b68aa0-b096-417c-b074-0f2d4b7e83e2"
filename	Filename	yes	string (Filename)	"tr100007_201910251210065_09.str" or "Pic100007_P001_L01_R1_XEX_D2019_09_04_T11_59_38_N00000021.jpg"
originalSender	Original sender of the file	yes	string (AutoPASS Actor ID)	"100007"
downloadTime	Timestamp for when the file was downloaded	yes	yyyy-MM-dd'T'HH:mm:ss (ISO 8601)	"2019-11-01T13:00:34"

Validation rules:

- originalSender must be equal to Actor ID in filename.

The JSON string shall also contain an array of error elements (one or more) - where each error shall refer to a specific line/record in the original file. The error element shall contain the following parameters:

Parameter name	Description	Mandatory value	Format	Sample
lineNumber	A line is a string in the text file that ends with a line separator. First line gets line number 1. This parameter will have no value in the case of a jpg file.	no	integer	23
chargingPoint	Number of the charging point where the passage has taken place	no *)	string (3 digits)	"065"
lane	Number of lane where passage took place	no *)	string (2 digits)	"02"
passageTime	Time when passage took place	no *)	yyyy-MM-dd'T'HH:mm:ss (ISO 8601)	"2019-11-01T13:00:34"
OBU	OBU id	no *)	Refer to 4.3 AutoPASS Data Formats.pdf, appendix A8	"9780030003120012 3A"
errorAtPosition	Start position for detected error. First character on a line has position 1. A value 'null' means that the error is not related to a specific position - it applies to the entire line (or file).	no	integer	130
errorCode	See table below	yes	integer in the range 100 to 999	178
errorDescription	Textual description of the error detected for the actual line	yes	string	"An error description"

*) The parameter shall correspond exactly to the parameter in the original tr file or picture file (filename). Refer to 4.3 AutoPASS Data Formats.pdf, appendix A8 and A9

Note on mandatory

All parameters shall be present in the JSON string. Missing values shall be represented as 'null' (without quotes).

Error codes (the list is not exhaustive and is subject to changes):

Error code	Error name	Description
General		
101	Duplicate	The same file has been processed already
Tr file		
201	Invalid file length	
202	Corrupt data	Use for example if the line is corrupted or not in line with the specification
203	Missing parameter value	
204	Invalid parameter value	Use also in case, for example the operator code, toll plaza or lane is unknown
205	Invalid parameter format	Use for example if the nationality does not comply with ISO 3166 Alpha 2
206	Match failure	Parameter does not match the value of the parameter in the filename
Picture file		
301	Unreadable picture	
302	Corrupt picture file	
303	Missing picture file	
304	Invalid parameter value	Filename parameter (for example charging point or lane number) is invalid
305	Match failure	Picture imposed parameters does not match with the filename

Samples:

Sample JSON for tr file


```
{
        "transactionId" : "51b68aa0-b096-417c-b074-0f2d4b7e83e2",
        "filename" : "tr100007_201910251210065_09.str",
        "originalSender" : "100007",
        "downloadTime" : "2019-11-01T13:00:34",
        "error" :
        [
            {
                "lineNumber" : 23,
                "chargingPoint" : "065",
                "lane" : "02",
                        "passageTime" : "2019-11-01T13:00:34",
                        "OBU" : "97800300031200123A",
                        "errorAtPosition" : 44,
                        "errorCode" : 204,
                        "errorDescription" : "The ServiceNumber is not numeric ("Ø~$")"
                },
                {
                "lineNumber" : 39,
                "chargingPoint" : "066",
                "lane" : "01",
                        "passageTime" : null,
                        "OBU" : "97800300031200124A",
                        "errorAtPosition" : 8,
                        "errorCode" : 203,
                        "errorDescription" : "Missing passage time"
            },
                {
                "lineNumber" : 99,
                "chargingPoint" : null,
                "lane" : null,
                        "passageTime" : null,
                        "OBU" : null,
                        "errorAtPosition" : null,
                        "errorCode" : 201,
                        "errorDescription" : "Line length (1024) is not correct (512)"
            }
    ]
}
```


Sample JSON for picture file


```
{
        "transactionId" : "51b68aa0-b096-417c-b074-0f2d4b7e83e2",
        "filename" : "Pic100007_P001_L01_R1_XEX_D2019_09_04_T11_59_38_N00000021.jpg",
        "originalSender" : "100007",
        "downloadTime" : "2019-11-01T13:00:34",
        "error" :
        [
            {
                "lineNumber" : null,
                "chargingPoint" : "001",
                "lane" : "01",
                        "passageTime" : "20190904115938000",
                        "OBU" : null,
                        "errorAtPosition" : null,
                        "errorCode" : 301,
                        "errorDescription" : "Unable to read picture - snow storm"
                }
    ]
}
```


HTTP status codes

The reject endpoint may return the following http status codes:

Code	Name	Description	Error handling
200	OK	Success	
400	Bad request	The JSON / contents is not valid	Check error message
401	Unauthorized	Authorization token is not valid	Login to get a new token
410	Gone	The file does not exist	Do not retry