CTC Wallet – API description for locations

Version 1.3.4

Trusted Carrier GmbH & Co. KG
Breitenbachstraße 1
D-60487 Frankfurt
Telefon: +49 (0) 89 890 569 – 280
Internet: www.trusted-carrier.com

Versions

Version	Date	Type of change
1.3.4	23.02.2023	Updates to sendNotification method
1.3.3	06.12.2023	Switch to new CI
1.3.2	08.08.2023	Reformatted JSONs
1.3.1	26.07.2023	Conversion to .docx format
1.3	24.05.2023	Added information for JWT security mode
1.2	27.09.2022	Update for v2.0.3 (223) of the mobile app - Added sendDraft method - Added cancelDraft method - Added cancelCertificate method
1.1.1	25.04.2022	Updates (for v2.0.1 (194) of the mobile app) - Update to sendNotification method
1.1	09.04.2022	Updates (for v2.0.1 (188) of the mobile app) - Added identifyDriver method - Added sendCertificate method - Updates to sendNotification method - Added generateTempSignedPostURL method to upload files
1.0	01.07.2021	First official version

Table of contents

Overview

This document explains the methods of version 1 of the CTC Wallet API along 2 example processes.

Technical requirements

The external system requires an online connection and an open port for requests from the CTC backend as well as the ability to send requests to https://walletapi.trusted-carrier.com/api/restAPIs/v1.

Security mode

The mode of security can be chosen on the screen “API settings” in the “Admin” module of a CTC account.

- “Default”: This is the basic security setting, allowing authorization with a key.
    - URL: In this field, the endpoint of the external system can be specified. If this field has a value, the ERP key field must also be filled
    - ERP key: CTC will send the value entered here towards the external system, when communication is initiated by CTC
    - CTC key: CTC creates a key, that the external system needs to send in all requests when communication is initiated by the external system
- “JWT”: This extends the security with JWT tokens in the authentication header, which must be sent by both CTC and the external system in each request.
    - CTC JWT key: CTC will create a private/public keypair in RSA2048 format
    - ERP JWT key: The value entered here must be in RSA2048 & ASN.1 format Each location will be have security key generated by CTC

If JWT is enabled:

- Both location and CTC APIs will be required to send a authorization header with the value: **Bearer <JWT token>**
- JWT settings
    - RS512
    - payload
        - iss: sender URL
            - CTC will use one of the following: https://ctc.trusted-carrier.com/ (production system), https://ctc-staging.trusted-carrier.com/ (test system)
            - While “payload” is not a mandatory value, the external system is advised to provide it and it will be stored in the CTC logs
        - iat: unixtime seconds

        - Timestamp when the token was issued
        - Must be before current time
    - exp: unixtime seconds
        - Timestamp when the token expires
        - Maximum of 60 seconds into the future
- The request will be rejected with a 401 response if:
    - Authorization header is missing or malformed, and the account has JWT enabled
    - JWT isn't verifiable using the public key provided by location
    - JWT is expired

Example configuration & driver data

The methods of the CTC Wallet API are described over the course of this document. For the example data exchanges illustrating these processes, imagine a driver who wants to register at an industrial site using the CTC Wallet mobile app. Unless otherwise stated, the driver is using a personal CTC Wallet device.

The industrial site's configuration requires the driver to provide given name(s), last name, birthdate, nationality, spoken languages, telephone number and ADR licence information. In addition, the list of vehicle components and a reference number to identify the transport order are to be provided.

The driver has assembled this information in the CTC Wallet mobile app.

The driver "Max Mustermann" is born 01.01.1980, is of German nationality, and is speaking German (primary language) as well as English & French (additional languages). The phone number is +49 172 1234567. He has an ADR licence with number "000-1111111" which is expiring on 31.12.2021. He is driving the vehicle "C-TC1010", which is registered in Germany. A personal "singleuser" device is used. Biometrics are used for authentication.

In process 1, the driver creates a QR code containing all this information. The QR code is scanned by an external system user. The external system sends the QR code data including authorization information towards the CTC backend using the method *qrCode*, from which it is returned to the external system.

In process 2, the driver sends the online registration directly towards the external system (process 2).

For both processes, the external system user may trigger the authentication / validation processes (*identifyDriver* / *updateDriver* / *updateField*), The external system may also send notifications to the driver's mobile device using the *sendNotification* method.

The CTC Web frontend configuration determines whether a CTC Wallet profile is used for a QR code registration or for an Online registration.

Process type 1: QR Code registration

The QR code is produced in the CTC Wallet mobile app for processes, where a QR code is scanned by an external system user. The QR code contains the assembled information in conjunction with meta information.

Example QR code:

```
{
    "dm": "singleuser", // device mode that the driver is using
    "am": "biometrics", // authentication method that the driver is using
    "p": 27, // profile ID identifying the location
    "pv": 15, // profile version (incremented with every configuration update)
    "u": 146, // driver CTC ID
    "upi": “7000610”, // Unique process identifier of registration draft
    "pp": 3, // privacy policy version of the destination, that the driver agreed to
    "rid": "0ee5c421-0010-442d-af0b-21c4305c387b", // unique registration ID
    "di": "dc65aa18-4769-4c03-928b-
80818edcfe3e", // driver device ID, created during installation
    "bi": "ee5570bc-f430-4f27-88e1-
fe86317efd87", // unique driver biometrics ID, created during biometric linking in the mobile
app
    "c": 1625478564, // registration created-at timestamp (unixtime)
    "rd": 1625478564, // registration date as chosen by the driver (unixtime)
    "os": “a”, // Operating system (Android / iOS)
    "osv": “13”, // Operating system version
    "av": “418”, // Build number
    "mod": “SM418F”, // Device model number
    "la": “EN”, // App language
    "d": "IVVBp9pVBpuSipGKqxMYmQ7gYOibOZy4aO0N1WlseYV8LLGOH0t60PQtEAhwHt0uv5RdeFOsM50Hokj7nRvp
VZAYFOIBptpVkvu8lj44k69vRj9p5VsZdFuHnUz97KrLM45Bv+kDDsufxQtmUR6xhIYTN7fl9vi9ZmAAxkH3Q/6KYRDezX
gnWn4aeVsvyL7SDedIG5r5YG3ulPr2mukDnDZhCuAFNovs8iR3q63ndbrzTZSG2KNKgUpUuZa+n7A9", // encrypted
registration data
    "k": "adad71eb3e32996cc532f15c9f7c764843a10cc0e5343b7d39ca0eae74896e78", // public key (dr
iver)
    "ki": 2, // location key id
    "iv": "6ef50289efe7bfff78e0f05b699b1b63" //  AES-initialization vector
}
```

After the QR code is scanned, the method *qrCode* is called by the external system to send the QR code content along with the CTC public key to the CTC backend. There it is decrypted, parsed & validated.

The response to the external system contains validation information about driver and vehicles and can be used to execute further processes in the external system. The driver also receives the information, that the QR code was processed as a silent notification.

Request from external system toapi/restAPIs/v1/qrCode :

```
{
    "publicKey": "<ctc-generated-publickey>",
    "data": {
        "qrCode": "<QR Code as described above>"
    }
}
```

Response from CTC:

```
{
    "error_code": {
        "code": "0",
        "message": "Success"
    },
    "payload": {
        "type": "walletRegistration",
        "uid": 146,
        "am": "biometrics",
        "dm": "singleuser",
        "d": {
            "f": {
                "fn": {
                    "v": "Max",
                    "s": "m"
                },
                "ln": {
                    "v": "Mustermann",
                    "s": "m"
                },
                "dob": {
                    "v": "19800101",
                    "s": "m"
                },
                "n": {
                    "v": "DE",
                    "s": "m"
                },
                "pl": {
                    "v": "DE",
                    "s": "x"
                },
                "l": {
                    "v": [
```

```
                    "EN",
                    "FR"
                ],
                "s": "x"
            },
            "t": {
                "v": "00491721234567",
                "s": "x"
            },
            "an": {
                "v": "111-0000000",
                "s": "m"
            },
            "av": {
                "v": "20213212",
                "s": "m"
            }
        },
        "p": {
            "refno": {
                "v": "1234",
                "s": "x"
            }
        },
        "mv": {
            "t": "1",
            "lp": "C-TC1010",
            "s": "v",
            "n": "DE"
        },
        "v": [],
        "sc": {},
    },
    “w”: [],
    "pid": 39,
    "rid": "5e2012bb-ac04-4cf7-e9ec-60a933bccc78",
    "upi": "7000610",
    "identified": true
  }
}
```


Structure of the data (“payload”) sent from CTC

- uid: driver CTC user ID (optional; will be sent if driver is a registered user)
- type: data type (here: walletRegistration)
- dm: device mode (possible values: singleuser, multiuser, kiosk)
- am: authentication mode (possible values: biometrics, passcode, none)
- f: driver master data fields

        - v: value
        - s: status - m (missing), v (validated), e (expired), c (changed), x (no validation allowed)
    - p: profile specific fields (up to 5)
        - v: value
        - s: status - x (no validation allowed)
    - mv: motorised vehicle
        - t: type
        - lp: licence plate / container number
        - n: nationality - not available for containers
        - s: status - m (missing), v (validated), e (expired)
        - v: master data of the vehicle component (optional; will be sent if destination has activated this configuration)
    - v: vehicles (array of up to 5 vehicles; if there are no vehicles, empty array is sent)
        - t: type
        - lp: licence plate / container number
        - n: nationality - not available for containers
        - s: status - m (missing), v (validated), e (expired)
        - v: master data of the vehicle component (optional; will be sent if destination has activated this configuration)
    - sc: carrier selected by the driver
        - n: name
        - id: CTC company ID
    - upi: unique process identifier (only sent, if driver used a draft)
    - w: warnings (array of warnings; if there are no warning, empty array is sent)
    - rid: registration id - will be used to identify validation and to send notifications to the driver
    - pid: profileID - id of the location profile
    - identified: true, if user identity is validated

For more details about driver masterdata fields, vehicle types, error codes, and warning messages, refer to the annex of this document.

Process type 2: Online Registration

The driver can also generate an online registration, where no QR code is created by the driver. In this process, the data is directly sent from the driver's device to the CTC backend, and afterwards to the endpoint provided by the external system.

Data sent to external system endpoint:


```
{
    "publicKey": "external system-public-key",
    "type": "walletRegistration",
```

```
"payload": {
    "uid": 146,
    "am":"biometrics",
    "dm":"singleuser",
    "d": {
        "f": {
            "fn": {
                "v": "Max",
                "s": "v"
            },
            "ln": {
                "v": "Mustermann",
                "s": "v"
            },
            "dob": {
                "v": "19800101",
                "s": "v"
            },
            "n": {
                "v": "DE",
                "s": "v"
            },
            "pl": {
                "v": "DE",
                "s": "x"
            },
            "l": {
                "v": [
                    "EN",
                    "FR"
                ],
                "s": "x"
            },
            "t": {
                "v": "00491721234567",
                "s": "x"
            },
            "an": {
                "v": "111-0000000",
                "s": "v"
            },
            "av": {
                "v": "20213212",
                "s": "v"
            }
        },
        "p": {
            "refno": {
```

```
                    "v": "1234",
                    "s": "x"
                }
            },
            "mv": {
                "t": "1",
                "lp": "C-TC1010",
                "s": "v",
                "n": "DE"
            },
            "v": [],
            "sc": {},
        },
        "w": [],
        "pid": 39,
        "rid": "a320326d-88e9-4fb3-91e7-cde7178db5aa",
        "upi": "7000610",
        "identified": true
    }
}
```

Methods

Identify Driver

URI: api/restAPIs/v1/identifyDriver

Functional description

To confirm the identity of a driver in the CTC system, it is required to manually confirm it, e.g., at a counter, by visual confirmation based on a photo ID. The process is only available for drivers using biometrics. The driver produces a QR code in the mobile app, which always contains the user ID, biometrics ID and device ID.

Additionally, the QR code may also contain driver masterdata (depending on the configuration), which should be displayed after the QR code scan in the external system, so that the destination's employee can match this data with the photo ID information.

Upon successful confirmation, *identifyDriver* should be called to create/update an entry in the database containing the biometrics-related IDs as well as validate the driver masterdata.

Example data exchange

Request sent to CTCWallet

```
{
    "publicKey": "ctc-generated-publickey",
    "data": {
        "qrCode": {
            "u": "121", // driver's user ID
            "cid": "1", // certificate ID (optional)
            "ucid": "user_certificate_id", // unique process ID (optional)
            "di": "device_id", // device ID
            "bi": "biometrics_id", // biometrics ID
            "c": 1643368782.298, // timestamp of the QR code creation (unixtime)
            "d": {
                "f": { // driver masterdata fields, as per certificate configuration
                    "fn": "Max",
                    "ln": "Mustermann",
                    "dob": "1357776000",
                    "n": "DE"
                }
            }
        }
```

```
    }
}
```

Response from CTCWallet

```
{
    "error_code": {
        "code": "0",
        "message": "Success"
    }
}
```

Update Driver

URI: api/restAPIs/v1/updateDriver

Functional description

updateDriver works similar to *identifyDriver*, but is tied to a registration process. The process is also available for drivers who do not use biometrics.

If a driver is not authenticated in the CTC system during a registration process (e.g., because no biometrics are used), it is required to manually confirm the identity of the driver, e.g., at a counter, by visual confirmation based on a photo ID. After successful confirmation, *updateDriver* should be called.

The driver associated with the provided registration ID will be authenticated. The response to *updateDriver* contains the full registration information including the updated validation information.

updateDriver will validate the driver's identity and create/update an entry in the database which saves the driver ID as well the device ID and biometrics ID. If the driver uses the same device with the same biometrics, the JSON will already contain validation information in the next registration process.

updateDriver will also automatically validate (if provided):

- fn: first name(s) of the driver
- ln: last name
- dob: birthdate
- b: nationality

If the driver is not using biometrics, or has changed the device / biometrics, the process for authentication has to be repeated for every registration.

Example data exchange

Request sent to CTC Wallet:

```
{
    "publicKey": "ctc-generated-publickey",
    "data": {
        "registrationID": "5e2012bb-ac04-4cf7-e9ec-60a933bccc78"
    }
}
```

Response from CTC Wallet:

The response from CTC Wallet is structurally identical to the response after the *qrCode* is called. However, the fields “fn”, “ln”, “dob” and “n” are now sent back with status “validated”.

Excerpt:


```
...
"fn": {
    "v": "Max",
    "s": "v"
},
"ln": {
    "v": "Mustermann",
    "s": "v"
},
"dob": {
    "v": "19800101",
    "s": "v"
},
"n": {
    "v": "DE",
    "s": "v"
}
…
```

Update Field Validations

URI: api/restAPIs/v1/updateField

Functional description

updateField should be called when an information provided by the driver was confirmed, e.g., by matching the provided information with a physical document. The information from the driver associated with the provided registration ID will be validated or invalidated.

The value for “action” may be “validate” or “invalidate”.

In this example, the ADR licence number & expiration date were confirmed.

Example data exchange

Request sent to CTC Wallet:

```
{
    "publicKey": "ctc-generated-publickey",
    "data": {
        "registrationID": "<ctc-generated-registrationID>",
        "objects": [
            {
                "fieldID": "an",
                "action": "validate"
            },
            {
                "fieldID": "av",
                "action": "validate"
            }
        ]
    }
}
```

Response from CTC Wallet:

```
{
    "error_code": {
        "code": "0",
        "message": "Success"
    }
}
```

Send message to driver

URI: api/restAPIs/v1/sendNotification

Functional description

This method sends a push notification to a driver's device. The correct driver is selected via the registrationID.

The notification may consist of the following fields:

- fileURL (500): Link to a downloadable PDF / image file
- url (500): Link to a website
- qrCode (1000): This string will be displayed as a scannable QR code on the driver's device, e.g., to be presented at the next process station on-site
- qrCodeEncrypt: value "true" will convert the content to a QR code signed using the JWT configuration; value "false" (default) will convert the content of field "qrCode" into a QR code without changing the data
- qrCodeAdd: adds profile-specific information from the registration to the JWT payload, if such information was part of the registration
- qrCodeAuth: deprecated, do not use anymore
- content (360): This will be displayed as standard text
- headerText (360): This will be displayed as a colored header
- headerType: possible values
    - "error": headerText will be displayed in red & bold letters
    - "warning": headerText will be displayed in orange & bold letters
    - "ok": headerText will be displayed in green & bold letters
- content (360): information for the driver (notification level)
- registrationInformation (1000): information for the driver (stored on registration level)

The fields fileURL, url, qrCode, content and (headerText & headerType) are optional, but at least 1 has to be filled. headerText & headerType must always be sent in conjunction, or not at all.

The number in brackets is the maximum length of each string. If more characters are sent, the string will be truncated.

Example data exchange

Request sent to CTC Wallet:

```
{
    "publicKey": "ctc-generated-publickey",
    "data": {
        "registrationID": "<ctc-generated registration ID",
        "fileURL": "https://test.com/file.pdf",
```

```
        "url": "https://certificate.com/online-course",
        "qrCode": "this will be displayed in the QR code",
        "qrCodeAdd": [
            "time"
        ],
        "qrCodeAuth": "none",
        "qrCodeEncrypt": 1,
        "headerText": "Checkin successful.",
        "headerType": "ok",
        "content": "You may enter the site.",
        "registrationInformation": "Additional information"
    }
}
```


Response from CTC Wallet:


```
{
    "error_code": {
        "code": "0",
        "message": "Success"
    },
    "payload": {
        "registrationID": "<ctc-generated-registrationID>",
        "fileURL": "https://test.com/file.pdf",
        "url": "https://certificate.com/online-course",
        "qrCode": "this will be displayed in the QR code",
        "qrCodeAdd": [
            "time"
        ],
        "qrCodeAuth": "none",
        "qrCodeEncrypt": 1,
        "headerText": "Checkin successful.",
        "headerType": "ok",
        "content": "You may enter the site.",
        "registrationInformation": "Additional information"
    }
}
```

Upload File

URI: api/restAPIs/v1/generateTempSignedPostURL

Functional description

Create a presigned URL to upload a file. This method is built based on AWS specification.
If you are not familiar with it, please check:
https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/modules/_aws_sdk_s3_presigned_post.html

Request structure

- publicKey: security check
- data
    - key - file name
        - must be unique, recommendation: use a uuid generated name
    - mimeType: type of the file
        - must be one of: "application/pdf", "image/jpeg", "image/png", "image/jpg"
    - size: size of the file in bytes
        - max value 1 000 000 (1Mb)

Example data exchange

Request sent to CTCWallet

```
{
    "publicKey": "ctc-generated-publickey",
    "data": {
        "key": "tmp/sample.pdf",
        "mimeType": "application/pdf",
        "size": "141041"
    }
}
```

Response from CTCWallet

```
{
    "error_code": {
        "code": "0",
        "message": "Success"
    },
    "payload": {
        "url": "https://s3.eu-west-1.amazonaws.com/st-ctc0-dev",
        "fields": {
            "Key": "tmp/sample.pdf",
            "bucket": "st-ctc0-dev",
```

```
            "X-Amz-Algorithm": "AWS4-HMAC-SHA256",
            "X-Amz-Credential": "AKIAWVWTIUUSWVYUSSG6/20220106/eu-west-1/s3/aws4_request",
            "X-Amz-Date": "20220106T122809Z",
            "Policy": "eyJleHBpcmF0aW9uIjoiMjAyMi0wMS0wNlQxMjozMzowOVoiLCJjb25kaXRpb25zIjpbWyJ
jb250ZW50LWxlbmd0aC1yYW5nZSIsMTMzLDE0OV0sWyJlcSIsIiRDb250ZW50LVR5cGUiLCJhcHBsaWNhdGlvbi9wZGYiX
SxbImVxIiwiJENvbnRlbnQtRGlzcG9zaXRpb24iLCJhdHRhY2htZW50Il0seyJLZXkiOiJ0bXAvc2FtcGxlLnBkZiJ9LHs
iYnVja2V0Ijoic3QtY3RjMC1kZXYifSx7IlgtQW16LUFsZ29yaXRobSI6IkFXUzQtSE1BQy1TSEEyNTYifSx7IlgtQW16L
UNyZWRlbnRpYWwiOiJBS0lBV1ZXVElVVVNXVllVU1NHNi8yMDIyMDEwNi9ldS13ZXN0LTEvczMvYXdzNF9yZXF1ZXN0In0
seyJYLUFtei1EYXRlIjoiMjAyMjAxMDZUMTIyODA5WiJ9XX0=",
            "X-Amz-
Signature": "b4a480044b147df3984582b3f672d6811c826c6ac22b019537f477ef43ddbd9f"
        }
    }
}
```


Steps for uploading file:

- create a pre-signed URL calling api/restAPIs/v1/generateTempSignedPostURL
- upload the file to S3 (format for curl sample)
  curl --location --request POST **{{data.payload.url}}**\ - from pre-signed response
  --form 'key=**{{data.payload.fields["Key"]}}**' \ - from pre-signed response
  --form 'X-Amz-Credential=**{{data.payload.fields["X-Amz-Credential"]}}**' \ - from pre-signed response
  --form 'X-Amz-Signature=**{{data.payload.fields["X-Amz-Signature"]}}**' \ - from pre-signed response
  --form 'Content-Type="**application/pdf**"' \ - must match pre-signed request
  --form 'Policy=**{{data.payload.fields["Policy"]}}**' \ - from pre-signed response
  --form 'X-Amz-Algorithm=**{{data.payload.fields["'X-Amz-Algorithm"]}}**' \ - from pre-signed response
  --form 'X-Amz-Date=**{{data.payload.fields["X-Amz-Date"]}}**' \ - from pre-signed response
  --form 'Content-Disposition="**attachment**"' \ - must be set to this specific value
  --form 'file=..." - file that is uploaded
- call api/restAPIs/v1/sendNotification, in the fileURL field send key (ex: tmp/sample.pdf)

Send Certificate

URI: api/restAPIs/v1/sendCertificate

Functional description

The location sends information about a certificate to CTC that will be pushed towards the driver. CTC stores this information to be able to later identify individual drivers based on the certificate content.

Request structure

- publicKey: security check
- data
    - driverID: id of the driver (corresponds to u key in the qrCode)
    - certificateID: id of the certificate (corresponds to cid key in the qrCode)
    - userCertificateID: id of the user certificate (corresponds to ucid key in the qrCode) - optional
    - certificateTTL: time until expiration is seconds, optional unless certificate configuration requires it
    - certificateData: content of the certificate (shortcode + values), shortcodes depend on certificate settings

Example data exchange

Request sent to CTCWallet

```
{
    "publicKey": "ctc-generated-publickey",
    "data": {
        "driverID": 121,
        "certificateID": 1,
        "userCertificateID": "user_certificate_id",
        "certificateTTL": 0,
        "certificateData": {
            "exam1": "<example_data>"
        }
    }
}
```

Response from CTCWallet

```
{
    "error_code": {
        "code": "0",
        "message": "Success"
```

```
    }
}
```

Cancel Certificate

URI: api/restAPIs/v1/cancelCertificate

Functional description

The location cancels an issued certificate. Cancellation is based on the ID of the user certificate, which was in the “userCertificateID” field of the QR Code during certificate creation.

Request structure

- publicKey: security check
- data
    - userCertificateID: id of the user certificate

Example data exchange

Request sent to CTCWallet

```
{
    "publicKey": "ctc-generated-publickey",
    "data": {
        "userCertificateID": "user_certificate_id"
    }
}
```

Response from CTCWallet

```
{
    "error_code": {
        "code": "0",
        "message": "Success"
    }
}
```

Send Draft

URI: api/restAPIs/v1/sendDraft

Functional description

The location sends information about a registration to CTC that will be pushed towards the driver. CTC stores information and this will be used as a base for a new online registration/QR code.

Request structure

- publicKey: security check
- data
    - editable: boolean
        - allow driver to edit all data of the draft
        - must be true for this version (false will result in an API error)
    - upi: string - unique process id
        - Allows relating the registration to a specific process in the external system
    - driverID: id of the driver
    - profileID: id of the profile that will be used
        - if not sent and the location has a single active profile then that profile will be used
    - time: number - unixtime - optional
        - date the registration will take place
    - carrierID - number - optional
        - ID of the carrier
        - if the driver is not associated with the carrier this will result in an API error
    - details - object - part of the registration that will be created
        - mv - object - data on the motor vehicle
            - lp - string - license plate
            - n - string - nationality
            - tid - string - vehicle type ID - optional if the vehicle is known in CTC Asset
        - v - list - list of trailers / containers
            - lp - string - license plate / container number
            - n - string - nationality
            - t - string - vehicle type ID - optional if the vehicle is known in CTC Asset
        - p - object - profile specific fields
            - f1, f2, ...other fields - up to 5

Note that all vehicle components must exist in CTC Asset, if no vehicle type ID is provided. If the vehicle component exists in CTC Asset, the type from the draft will be ignored. If the vehicle does not exist in CTC Asset, the type from the draft will be used. The vehicle components must form a valid vehicle set in CTC Wallet, otherwise sending the draft will result in an API error.

Example data exchange

Request sent to CTCWallet


```
{
    "publicKey": "ctc-generated-publickey",
    "data": {
        "editable": true,
        "upi": "test2-1",
        "driverID": 121,
        "profileID": 145,
        "time": 1658134431,
        "carrierID": 10,
        "details": {
            "mv": {
                "lp": "C-TC1010",
                "n": "DE"
            },
            "v": [
                {
                    "lp": "C-TC1015",
                    "n": "DE"
                },
                {
                    "lp": "CLUU1234567"
                }
            ],
            "p": {
                "s1": "testing2"
            }
        }
    }
}
```


Response from CTCWallet


```
{
    "error_code": {
        "code": "0",
        "message": "Success"
    }
}
```

Cancel Draft

URI: api/restAPIs/v1/cancelDraft

Functional description

The location cancels a draft. Drafts can only be canceled, if the driver has not used the draft (identified by the “upi” value) in a registration yet.

Request structure

- publicKey: security check
- data
    - upi: string - unique process id

Example data exchange

Request sent to CTCWallet

```
{
    "publicKey": "ctc-generated-publickey",
    "data": {
        "upi": "unique_process_id"
    }
}
```

Response from CTCWallet

```
{
    "error_code": {
        "code": "0",
        "message": "Success"
    }
}
```

Update registration status

URI: api/restAPIs/v1/updateRegistrationStatus

Functional description

The external system sends an update about how the registration was processed. This is a way to quantify how effective the registration was.

The field “status” is an integer value. Current possible values are:

- 0: to indicate that something went wrong.
- 10 or 20: to indicate good cases

The field “details” is optional. It can contain human-readable information, which may be particularly helpful in error cases.

Example data exchange

Request sent to CTCWallet

```
{
    "publicKey": "<ctc-generated-registrationID>",
    "data": {
        "registrationID": "the ID of the registration",
        "status": 10
    "details": "Human-readable information."
    }
}
```

Response from CTCWallet

```
{
    "error_code": {
        "code": "0",
        "message": "Success"
    }
}
```

Annex

List of warnings

Warning	Scenario
profile_version_updated	User generated a registration with an outdated location profile version
profile_missing_encryption	QR code has no encryption, but the location requires it. Or QR code has encryption, but the location has it disabled.
biometrics_changed	User changed the biometric information in the mobile device. Server will not allow field validations until the device is validated
device_changed	User changed the device OR user is new. Server will not allow field validations until the device is validated
biometrics_missing	User biometrics is disabled
unknown_driver	User is using the “Guest mode”.

List of error codes

Code	Message	Description
0	Success	No error
4000	Missing Parameters	A mandatory parameter is missing
4001	Invalid Parameters	A parameter is incorrect, e.g., expecting number and received string
5001	QRCode expired	QR code has a 30 sec lifespan. After this, the API will not decode it
5002	QRCode invalid	QR code structure is corrupted, e.g., because of an incomplete JSON
5003	Registration expired	Registrations expire after 7 days. After this, the API can not process methods for this registration ID
5004	Device has changed	If a field validation is sent while the driver is not using a validated device, e.g., new user, or device change. Validations are only possible for validated devices.
5005	Invalid field or value	A field is not recognised, because it does not exist in the database
9004	Invalid User	Invalid user, e.g., because the public key is unknnown

List of vehicle types

Motorized vehicles		Non-motorized vehicles		Containers & swap bodies	
ID	**Type**	**ID**	**Type**	**ID**	**Type**
100	Semi-truck	201	Trailer (Chassis) 20'	301	Box Container 20'
111	Truck (Chassis) 20'	202	Trailer (Chassis) 30'	302	Box Container 30'
112	Truck (Chassis) 30'	203	Trailer (Chassis) 40'	303	Box Container 40'
121	Truck (Curtain)	211	Trailer (Curtain)	304	Box Container 45'
122	Truck (Box)	212	Trailer (Box)	311	Tank Container 20'
123	Truck (Thermo)	213	Trailer (Thermo)	312	Tank Container 30'
124	Truck (Tank)	214	Trailer (Tank)	313	Tank Container 40'
125	Truck (Silo)	215	Trailer (Silo)	314	Tank Container 45'
126	Truck (Gas)	216	Trailer (Gas)	321	Silo Container 20'
127	Truck (open)	217	Trailer (open)	322	Silo Container 30'
130	Car	221	Semi-trailer (Chassis) 20'	331	Gas Container 20'
140	Small transporter	222	Semi-trailer (Chassis) 30'	332	Gas Container 30'
		223	Semi-trailer (Chassis) 40'	340	Swap body
		231	Semi-trailer (Curtain)	350	Dump body
		232	Semi-trailer (Box)		
		233	Semi-trailer (Thermo)		
		234	Semi-trailer (Tank)		
		235	Semi-trailer (Silo)		
		236	Semi-trailer (Gas)		
		237	Semi-trailer (open)		

Driver masterdata fields

Short code	Field name	Format in JSON	Can be validated
fn	Given name(s)	Max. 40 characters	Yes
ln	Family name	Max. 40 characters	Yes
dob	Birthdate	8 numbers, YYYYMMDD	Yes
n	Nationality	2-character ISO code, in capitals	Yes
pl	Primary language	2-character ISO code, in capitals	No
l	Additional languages	Collection of 2-character ISO codes, in capitals	No
t	Telephone	Numbers only, max. 20 characters	No
lc	(C) Validity driving licence	8 numbers, YYYYMMDD	Yes
lce	(CE) Validity driving licence	8 numbers, YYYYMMDD	Yes
lc1	(C1) Validity driving licence	8 numbers, YYYYMMDD	Yes
lc1e	(CE) Validity driving licence	8 numbers, YYYYMMDD	Yes
an	ADR licence number	Max. 20 characters	Yes
aa	ADR licence addon courses	Collection with options: "TA" (tanks), "C1" (class 1), "C7" (class 7)	Yes
av	ADR licence validity	8 numbers, YYYYMMDD	Yes
in	ISOPA licence number	Max. 20 characters	Yes
iv	ISOPA licence validity	8 numbers, YYYYMMDD	Yes