Alexela OCC Partners API

This document describes Alexela OCC's (online card center's) online data exchange protocol for Alexela partners.

Table of contents

API protocol

REST/JSON protocol are used for the API. Character set in all communication and data content must be in UTF-8 encoding. All dates and date times must be in UTC.

Test and production system

Test service can be found: *https://test-partnerapi.alexela.ee/api/v1...*

Swagger: https://test-partnerapi.alexela.ee/swagger/index.html

Production service can be found: *https://partnerapi.alexela.ee/api/v1...*

Swagger: https://partnerapi.alexela.ee/swagger/index.html

Authentication

Every request is authenticated with a calculated signature sent as a HTTP header that is based on the request, current time and pre-shared client key (PSK).

A canonical string is first created using HTTP headers containing the content-type, content MD5 sum, request URI, and the timestamp.

`canonical_string = 'content_type,content_md5,request_uri,timestamp'`

Described parts are joined by commas.

content_type: Content Type used in request. Is always *application/json*

content_md5: MD5 checksum of the request content as hexadecimal string.

request_uri: Request uri for the given request starting with the /api/ root path component including the query string if applicable, e.g. */api/v1/getBalance*

timestamp: HTTP-date defined by RFC 2616

This string is then used to create the signature which is as Base64 encoded SHA256 HMAC calculated using the PSK.

The signature is then added as the ***Authorization*** HTTP header as follows

`APIAuth-HMAC-SHA256 $client_access_id:$signature`

On the server side, the SHA256 HMAC is computed in the same way using the request headers and the client's PSK, which is only know to the client and server but can be looked up on the server using the client's access id that is attached in the header. The access id is unique for each ERP and also identifies the calling ERP to the API.

Once generated the signature is valid for 5 minutes relative to API server time to avoid replay attacks so the client must ensure correct time is sent. The PSK must be at least 128 bytes of cryptographically secure random hex characters.

Example of pre-requisite script that works in postman if you change appId and APIKey to your correct data

```
var AppId = "******";
var APIKey = "*****";

var currentDate = new Date();
//var requestURI = "/" +
pm.environment.values.substitute(pm.request.url, null,
false).getRaw().split('/').splice(3).join('/');
//requestURI = requestURI.split("?")[0];

var requestPath = "/" + pm.request.url.path.join("/");  //"/" +
pm.environment.values.substitute(pm.request.url, null,
false).getRaw().split('/').splice(3).join('/');
var requestQueryString = pm.request.url.getQueryString();
var requestURI = requestPath + (!!requestQueryString ? '?' +
requestQueryString : '');

console.log(requestURI)

var requestContentBase64String = "";
if (pm.request.body && pm.request.body[0]) {
    var md5 = CryptoJS.MD5(pm.request.body.toString());
    //requestContentBase64String = CryptoJS.enc.Base64.stringify(md5);
    requestContentBase64String = md5.toString();
}

var signatureRawData =
`application/json,${requestContentBase64String},${requestURI},${curren
tDate.toUTCString()}`; //check
var signature = CryptoJS.enc.Utf8.parse(signatureRawData);
var secretByteArray = CryptoJS.enc.Base64.parse(APIKey);
var signatureBytes = CryptoJS.HmacSHA256(signature, APIKey);
var requestSignatureBase64String =
CryptoJS.enc.Base64.stringify(signatureBytes);
```

```

var hmacKey = "APIAuth-HMAC-SHA256 " + AppId + ":" +
requestSignatureBase64String;
var hmacDate = currentDate.toUTCString();

console.log(AppId, APIKey, hmacDate, hmacKey)

// pm.variables.set("hmacDate", currentDate.toUTCString());
// pm.variables.set("hmacKey", hmacKey);

pm.request.headers.add({ key: "Date", value: hmacDate });
pm.request.headers.add({ key: "Authorization", value: hmacKey });

pm.request.headers.add({ key: "Content-Type", value:
"application/json" });


var crmKvp = pm.request.url.query.find(x => x.key ==
"crm_account_id");
console.log(crmKvp, pm.request.url.query);
if (!!crmKvp) {
    pm.request.headers.add({ key: "crm_account_id", value:
crmKvp.value });
}
```

Additional headers

In addition to the Authorization header, the ***Date*** header must be set with the same timestamp value that was used in the HMAC calculation (for testing purposes the example pre-requisite header is already set properly)

Failed requests

Any returned HTTP status code above or equal to 400 should considered an error. All 4xx codes are client errors where the client must fix the request before resending it to the server. 5xx codes are server errors indicating that there is an error on the server side.

Example:

```
POST /api/v1/ContractTransactions HTTP/1.1
Content-Type: application/json
{
  "contractNumbers": [
    "string"
  ],
  "fromDate": "string",
  "toDate": "string"
}

HTTP/1.1 400 Bad Request
Content-Type: application/json

{
  "success":false,
  "data":null,
  "error":"Wrong Date format",
  "message": null
}
```

Postman pre-request script

```
var AppId = "your-api-id";
var APIKey = "your-api-key";

var currentDate = new Date();
```

```

var requestPath = "/" + pm.request.url.path.join("/");
var requestQueryString = pm.request.url.getQueryString();
var requestURI = requestPath + (!!requestQueryString ? '?' +
requestQueryString : '');

var requestContentHexString = "";
if (pm.request.body) {
    var md5 = CryptoJS.MD5(pm.request.body.toString());
    requestContentHexString = md5.toString(CryptoJS.enc.Hex);
}

var signatureRawData =
`application/json,${requestContentHexString},${requestURI},${currentDa
te.toUTCString()}`;
var signature = CryptoJS.enc.Utf8.parse(signatureRawData);
var signatureBytes = CryptoJS.HmacSHA256(signature, APIKey);
var requestSignatureBase64String =
CryptoJS.enc.Base64.stringify(signatureBytes);
var hmacKey = "APIAuth-HMAC-SHA256 " + AppId + ":" +
requestSignatureBase64String;
var hmacDate = currentDate.toUTCString();

pm.variables.set("hmacDate", hmacDate);
pm.variables.set("hmacKey", hmacKey);

console.log('uri', requestURI, 'string to tokenize', signatureRawData,
'hmac token', hmacKey);
```

Use variables `hmacDate` and `hmacKey` to set headers.

Payment card operations

Contract numbers request

GET /ContractNumbers

Operation allows to inquire contract numbers that are allowed to be seen by the specific API user.

Contract numbers – API user mappings are managed in Alexela OCC.

Response

Field	Type	Description
	Array(string)	List of contract numbers

```
[
  "string"
]
```

Transactions request

POST /ContractTransactions or GET/ ContractTransactions

Operation enables to inquire payment card transactions data for the selected contracts and time period.

POST Request

Field	Type	Description

contractNumbers	Array (string)	List of contract numbers
fromDate	Datetime	For example „2018-09-05“
toDate	Datetime	For example „2018-09-06“

```
{
  "contractNumbers": [
    "string"
  ],
  "fromDate": "string",
  "toDate": "string"
}
```

GET Request

https://{baseUrl}/ContractTransactions?contractNumbers=22297,123,123&fromDate=2022-07-01&toDate=2022-07-07

Response

Field	Type	Description
	Array(transactionRow)	List of transactions

TransactionRow:

Field	Type	Description
transactionId	String	
terminalId	String	
stationId	String	
receiptNo	String	
authorizationCode	String	Value generated by payment terminal
cardNumber	String	
currency	String	EUR, SEK, PLN etc
occProductCode	String	
occProductName	String	
occProdyctType	String	Balance transfer, Fuel, Insurance, Merchandise, Package, Service, Stamps

plateNumber	String	
driverName	String	
statusDate	Datetime	
invoiceDate	Datetime	Exists after the transaction has been invoiced
invoiceNo	String	Exists after the transaction has been invoiced
transactionTime	Datetime	
quantity	Decimal(18,4)	
netPricePerUnit	Decimal(18,4)	
discountPerUnit	Decimal(18,4)	
pricePerUnit	Decimal(18,4)	
totalPrice	Decimal(18,4)	
vatRate	Decimal(18,4)	
vatAmount	Decimal(18,4)	
netAmount	Decimal(18,4)	
odometerReadout	Integer	


```
[
  {
    "transactionId": "string",
    "terminalId": "string",
    "stationId": "string",
    "receiptNo": "string",
    "authorizationCode": "string",
    "cardNumber": "string",
    "currency": "string",
    "occProductCode": "string",
    "occProductName": "string",
    “occProductType”: “string”
    "plateNumber": "string",
    "driverName": "string",
    "statusDate": "2020-02-26T14:02:36.994Z",
    "invoiceDate": "2020-02-26T14:02:36.994Z",
    "invoiceNo": "20810601-19091",
    "transactionTime": "2020-02-26T14:02:36.994Z",
    "quantity": 0,
    "netPricePerUnit": 0,
    "discountPerUnit": 0,
    "pricePerUnit": 0,
    "totalPrice": 0,
    "vatRate": 0,
    "vatAmount": 0,
    "netAmount": 0,
    "odometerReadout": 0
  }
```

```
]
```

Possible error messages

- Wrong date format
- Only 60 days of search at one time is supported
- Only search over last 5 months of transactions is supported

Bonuscard operations

Define bonuscards request

POST /DefineBonuscards

Possible to add bonuscards to regular drivers group or premium drivers group. Based on the grouping, bonuscards will receive different discounts in Alexela stations.

Discounts for the groups are managed and controlled in Alexela OCC (discount settings, products, values, etc).

Request

Field	Type	Description
bonuscards	Array (string)	Define the list of regular bonuscards
premiumBonuscards	Array (string)	Define the list of premium bonuscards (with extra discount)

```
{
  "bonuscards": [
    "string"
  ],
  "premiumBonuscards": [
    "string"
  ]
}
```

Response

In case of successful POST request – OK response.

Bonuscard transactions request

POST /GetBonuscardTransactions

Operation enables to inquire transactions of partner bonuscards (the bonuscards that are previously defined with operation POST /DefineBonuscards).

Request

Field	Type	Description

fromDate	Datetime	From, for example „2018-09-05“
toDate	Datetime	To, for example „2018-09-06“

Response

Field	Type	Description
transactionId	String	
terminalId	String	
stationId	String	
receiptNo	String	
authorizationCode	String	
cardNumber	String	
currency	String	EUR, SEK, PLN etc
occProductCode	String	
occProductName	String	
statusDate	Datetime	
transactionTime	Datetime	
quantity	Decimal(18,4)	
netPricePerUnit	Decimal(18,4)	
discountPerUnit	Decimal(18,4)	
pricePerUnit	Decimal(18,4)	
totalPrice	Decimal(18,4)	
vatRate	Decimal(18,4)	
vatAmount	Decimal(18,4)	
netAmount	Decimal(18,4)	
plateNumber	String	
driverName	String	
odometerReadout	Integer?	


```
[
  {
    "transactionId": "string",
    "terminalId": "string",
    "stationId": "string",
```

```
    "receiptNo": "string",
    "authorizationCode": "string",
    "cardNumber": "string",
    "currency": "string",
    "occProductCode": "string",
    "occProductName": "string",
    "statusDate": "2020-02-26T14:21:10.558Z",
    "transactionTime": "2020-02-26T14:21:10.558Z",
    "quantity": 0,
    "netPricePerUnit": 0,
    "discountPerUnit": 0,
    "pricePerUnit": 0,
    "totalPrice": 0,
    "vatRate": 0,
    "vatAmount": 0,
    "netAmount": 0,
    "plateNumber": "string",
    "driverName": "string",
    "odometerReadout": 0
  }
]
```


Possible error messages

- Wrong date format
- Only 60 days of search at one time is supported
- Only search over last 5 months of transactions is supported

EV transactions operations

EV transactions request

GET /EvTransactions?fromDate=2023-12-01&toDate=2023-12-31

Operation enables to inquire EV transactions for specified date range. EV transactions query must be specifically enabled for a partner.

Response

Field	Type	Description
	Array(EvTransactionRow)	List of EV transactions

EvTransactionRow:

Field	Type	Description
transactionId	Integer	
stationId	Integer	
chargerId	String	Specific machine that was used for charging
transactionStatus	String	The status of the transaction. Possible valus: “confirmed“,

		“unconfirmed“, “cancelled“
transactionTime	DateTime	
transactionStartTime	DateTime	Start of charging
quantity	Decimal	kWh
connectorNumber	Integer?	The connector number of the charger that was used for charging

Possible error messages

- Wrong date format
- Only 60 days of search at one time is supported
- Only search over last 5 months of transactions is supported