Table Tap Go

Server Control API

Specification

Rev. 1.2

TTG Server Version 1.44.20.8

Content

Revision history

Rev.	Date	Author	Description
0.1	14.05.2019	AB	First release
0.2	21.10.2019	AB	Added “getkeglevels”, “getkeglevelcount” requests
0.3	21.11.2019	AB	Modified “productlist”, “addproduct”, “changeproductprice”. Added: “firmwareupdate”, “timesyncAll”, “restartAll”, “generateexportconfigdb”, “generateviewsdb”, “databasebackup”, “optimizedb”,
0.4	11.12.2019	AB	Added: currentcards, transactions, cardpurchases, serverconfig, setserverconfig
0.5	10.08.2020	AB	Added: gethappyhours, changehappyhour, deletehappyhour, remoteLogin, remoteLoginPurchase, getPortionsForDeviceAndLine
0.6	09.02.2021	AB	Added: restartclientapplication, getdevicecount, getdevicestatus, setbacklightbrightness, getserverbasicinfo, getdrivinglicencedata, Updated several endpoints to new data (getkeglevel, etc.)
0.8	17.08.2021	AB	Gettimestampconfigexport added
0.9	22.09.2021	AB	Cardpurchase, cardinfo added
1.0	04.04.2022	AB	Deposit and alcohol_allowed flag added for cardpurchase request
1.1	12.04.2023	AB	Transaction endpoint added (for adding transactions)
1.2	19.04.2023	AB	Documentation errors corrected for endpoints cardinfo + transaction

1 General

The purpose of this document is to provide a description of connecting to the TTG server via the server control API to e.g. change products, get information from the server, etc.

1.1 Benefit

The given API makes it possible to perform changes in the TTG server database and distribute the changed information to the TTG clients without the need of using the provided web interface (e.g. for 3rd party applications)

2 Overview

2.1 Common system setup

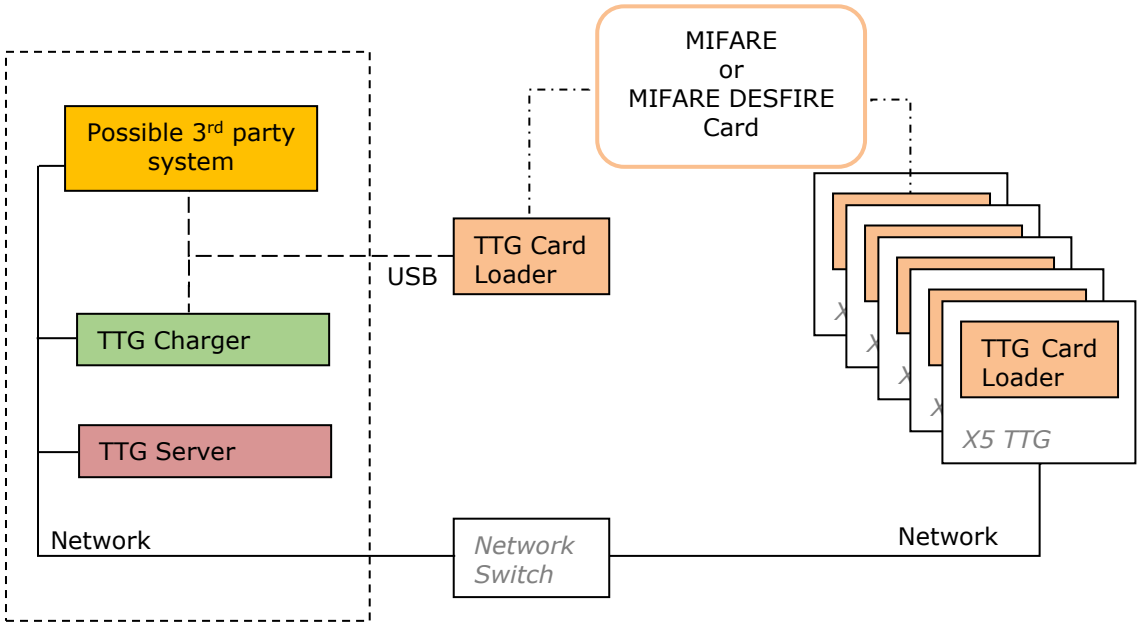
The system contains the X5 TTG clients (our dispenser machines) and some kind of computer equipment, including the TTG Server and a TTG Charger. The 3rd party system (e.g. a POS) can run on the same Windows machine. All parts are connected via network interfaces.

The TTG Charger can be used to create and balance cards etc.

The web interface of the TTG server is usually used for configuration purposes (product database etc.) as well as maintenance tasks (e.g. blocking lost system cards if administrated by the TTG system), but with the TTG server control API configuration changes can also be done via this API.

Booked transactions will be transferred to the TTG server. After successful transfer the booking record will be deleted from the TTG client.

This setup uses our MIFARE DESFIRE cards and thus our TTG Card Loaders can be used.



3 API description

The TTG server control API is a set of HTTP request/response messages with JSON payload.

The definition of the message structures only shows relevant data.

Values stated between <> are variable parameter fields.

3.1 General

All communication takes place over network links using HTTP protocols.

Access is granted by JWT (JSON Web Token) using the Bearer schema (Bearer Token)

The TTG server will respond with HTTP status "403 Forbidden" if an invalid authentication token is used.

If the TTG server responses with "500 Internal Server Error" (e.g. because the server cannot store the received data) the API-client shall retry its request.

3.2 Authentication

Usage

Request the authentication (JWT bearer) token from the TTG server.

Request from client

HTTP POST request to *<server IP address>*:*<HTTP port>*:

No authentication

```
POST /m2m/api/authtoken
Content-Type: application/json
Host: <server IP address>:<HTTP port>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
JSON payload	"username": specified user name (string) "password": specified password (string) "id": ID of the API-client (int64) "name": name of the API client (string) "type": json-server-control (string) – defines the type of the API user "version": 1

Tipp: ID and name can be chosen freely. Just make sure not to use any ID or name twice on the same TTG server to be able to distinguish between them more easily.

Allowed types:

Type	*Description*
json-server-control	API users can execute every API call for controlling the TTG server described in this document

Response from server

HTTP response to *<client IP address>*:*<HTTP port>*:

No authentication

```
HTTP 400 Bad request
Content-Type: application/json

HTTP 400 – Bad request
```

JSON payload	"error": username and/or password wrong

Authenticated

```
HTTP 200 OK
Content-Type: application/json

HTTP 200 - OK
```

JSON payload	"authtoken": 62 char string to use for further calls (string) „uuid": uuid of server (string)

3.3 Send configuration update notifcation

Usage

Create an export config database and send it to all connected clients

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/configupdate
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.4 Get productlist from server

Usage

Get a list of all configured products in the TTG server database

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/productlist
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "plu": plu of a specific product (optional) – if you just want the information about 1 specific product

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload		„uuid": uuid of server (string)
	JSON array "productlist"	"plu": plu auf product (int) "is_active": if product is marked as active or not (int) 1-yes 0-no "is_in_use": if product is in use at least at one client (int) 1-yes 2-no "is_advert": currently not used "volume_unit": volume unit of product (string) "volume_unit_dp": decimal places for volume (int) "volume_base_unit_divider": divider to main volume unit configured at server (int) "price_per_unit": price per unit in cents (fixed 2 decimal points) "price_per_unit_happyhour1": price per unit in cents for happyhour1 "price_per_unit_happyhour2": price per unit in cents for happyhour2 "price_per_unit_happyhour1_percent": price discount in % for happyhour 1 "price_per_unit_happyhour2_percent": price discount in % for happyhour 2 "units_per_serving": how many 'alcohol' units are used per serving. Needed for responsibility limit. (int) "name" name of product (string) "tasting_notes": detailed description of product (string) "brewery": name of brewery (string) "style": style of product (string) "abv": -1 = no abv – all other values > -1 set the ABV active (int) "ibu": -1 = no IBU – all other values > -1 set the IBU active (int) "product_type": (int) 0 – unknown 1 – beer 2 – wine 3 – cocktail

		4 – non-alcoholic 5 – games 6 - food

3.5 Get portion list from server

Usage

Get a list of all configured portions in the TTG server database

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/portionlist
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload		„uuid": uuid of server (string)
	JSON array "portionlist"	"portion_name": name of portion (string) "quantity": volume amount of portion "quantity_dp": volume amount decimal points

3.6 Get itemlist from server

Usage

Get a list of all configured items in the TTG server database

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/itemlist
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload		„uuid": uuid of server (string)
	JSON array "itemlist"	"item_id": id of the item (string) "product_plu": plu number of product used for item (int) "portion_name": name of the portion used for item (string) "price": price for item (int) "price_dp": decimal points used for price (int) "price_happyhour1": price for happyhour1 "price_happyhour1_dp": decimal points used for price (int) "price_happyhour2": price for happyhour2 "price_happyhour2_dp": decimal points used for price (int) "price_happyhour1_percent": percentage discount (int) "price_happyhour1_percent_dp": decimal points used for percentage discount (int) "price_happyhour2_percent": percentage discount (int) "price_happyhour2_percent_dp": decimal points used for percentage discount (int)

3.7 Edit an existing product

Usage

Edit an already existing product in the database

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/editproduct
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "plu": plu auf product (int) "is_active": if product is marked as active or not (int) 1-yes 0-no "is_advert": currently not used "volume_unit": volume unit of product (string) "volume_unit_dp": decimal places for volume (int) "volume_base_unit_divider": divider to main volume unit configured at server (int) "price_per_unit": price per unit in cents (fixed 2 decimal points) "price_per_unit_happyhour1": price per unit in cents for happyhour1 "price_per_unit_happyhour2": price per unit in cents for happyhour2 "price_per_unit_happyhour1_percent": price discount in % for happyhour 1 "price_per_unit_happyhour2_percent": price discount in % for happyhour 2 "units_per_serving": how many 'alcohol' units are used per serving. Needed for responsibility limit. (int) "name" name of product (string) "tasting_notes": detailed description of product (string) "brewery": name of brewery (string) "style": style of product (string) "abv": -1 = no abv – all other values > -1 set the ABV active (int) "ibu": -1 = no IBU – all other values > -1 set the IBU active (int) "product_type": (int) 0 – unknown 1 – beer 2 – wine 3 – cocktail 4 – non-alcoholic 5 – games 6 - food

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.8 Change the product logo of an already existing product

Usage

Change the product logo of an already existing product

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/changeproductlogo
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "plu": plu auf product (int) "product_logo": base64 encoded image (only .png and .jpg allowed)

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

Image to big

```
HTTP 200 – OK
```

JSON payload	"resize-information": "image has been resized to: optwidth: 391 and optheight: 231"

3.9 Add a product

Usage

Add a new product to the product database

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/addproduct
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "is_active": if product is marked as active or not (int) 1-yes 0-no "is_advert": currently not used "volume_unit": volume unit of product (string) "volume_unit_dp": decimal places for volume (int) "volume_base_unit_divider": divider to main volume unit configured at server (int) "price_per_unit": price per unit in cents (fixed 2 decimal points) "price_per_unit_happyhour1": price per unit in cents for happyhour1 "price_per_unit_happyhour2": price per unit in cents for happyhour2 "price_per_unit_happyhour1_percent": price discount in % for happyhour 1 "price_per_unit_happyhour2_percent": price discount in % for happyhour 2 "units_per_serving": how many 'alcohol' units are used per serving. Needed for responsibility limit. (int) "name" name of product (string) "tasting_notes": detailed description of product (string) "brewery": name of brewery (string) "style": style of product (string) "abv": -1 = no abv – all other values > -1 set the ABV active (int) "ibu": -1 = no IBU – all other values > -1 set the IBU active (int) "product_type": (int) 0 – unknown 1 – beer 2 – wine 3 – cocktail 4 – non-alcoholic 5 – games 6 – food "plu": (optional): desired plu for product. If plu already exists the "overwrite" parameter has also to be set "overwrite": (optional): if you add a product with a desired plu and the plu already exists you can overwrite the existing product with this flag

Response from server
HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	"info": "product added to product database with plu: #plu-number#"

3.10 Change price of an existing product

Usage

Edit prices of an already existing product

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/changeproductprice
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "products": JSON array
JSON array	"plu": plu auf product (int) "price_per_unit": price per unit in cents (fixed 2 decimal points) "price_per_unit_happyhour1": price per unit in cents for happyhour1 "price_per_unit_happyhour2": price per unit in cents for happyhour2 "price_per_unit_happyhour1_percent": price discount in % for happyhour 1 "price_per_unit_happyhour2_percent": price discount in % for happyhour 2

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.11 Get keg levels

Usage

Get the keg levels from given device and line.

This request is mainly used by the TTG server itself for refreshing the keg level information on the keg management page

Request from client

HTTP POST request to *<server IP address>*:*<HTTP port>*:

```
POST /m2m/api/getkeglevels
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"device_id": id of TTG device (int) "line_num": line number of device (int)

Response from server

HTTP response to *<client IP address>*:*<HTTP port>*:

```
HTTP 200 – OK
```

JSON payload	„uuid": uuid of server (string) "device_id": id of TTG device (int) "line_num": line number of device (int) "fill_level_perc": fill level in percent with fixed 2 decimal points (int) "fill_level_keg_size": keg size (int) "fill_level_keg_size_dp": decimal points of keg size (int) "tapping_date": when was the keg changed at last (epoch time – int) "pricing_state": Which price is currently in effect for this keg (int) 0: default price 1: happyhour1 (absolute price) 2: happyhour2 (absolute price) 3: happyhour1percent (percent discount on default price) 4: happyhour2percent (percent discount on default price)

Example:

Request:

Respone:

```
{
    "uuid": "0f4b5b42-cd7b-11ea-b368-302432cff25f",
    "device_id": 216548272414517,
    "line_num": 1,
    "fill_level_perc": 0,
    "fill_level_keg_size": 0,
    "fill_level_keg_size_dp": 0,
    "tapping_date": 0,
    "pricing_state": 0
}
```


Fill level would be 0% in this example

3.12 Get keg level count

Usage

Get the information how many keg levels are currently stored on the TTG server.

This request is mainly used for the TTG server keg management page. If the count is changed the whole page refreshes.

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/getkeglevelcount
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"old_count": current number of keg levels (int)

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	"keglevelcount": current count of keg levels configured (int)

3.13 Send firmware update notification

Usage

Sends a firmware update notification to all connected clients from the TTG server

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/firmwareupdate
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.14 Send timesync request

Usage

Sends a timesync notification/request to all connected clients of the TTG server

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/timesyncAll
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.15 Send restart notification

Usage

Sends a restart notification/request to all connected clients of the TTG server

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/restartAll
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.16 Generate Export Config Database

Usage

Generates the export config database which can be sent to the clients in a later step

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/generateexportconfigdb
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.17 Generate Views Database

Usage

(Re-)Generates the views database which is used for reporting purposes

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/generateviewsdb
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.18 Database backup

Usage

Do a database backup from the server databases

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/databasebackup
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.19 Optimize databases

Usage

Does a "optimize" databases operation (analyze + vacuum) on the server databases config, data_raw and data_view

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/optimizedb
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.20 Get current cards

Usage

Get the current active customer cards and/or the current active system cards of the TTG system

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/currentcards
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "customer_cards": 1/0 (1 – yes, 0 – no) return customer cards or not "system_cards": 1/0 (1 – yes, 0 – no) return system cards or not

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload		„uuid": uuid of server (string)
	JSON array "customer_cards"	"card_id": id of card (int) "is_blocked": true/false (Boolean) "blocked_comment": comment for block (string) "card_type": see table card types * (string) "customer_name" name of customer (string) "volume_consumed": already consumed volume (float) "volume_limit": volume limit for customer (float) "until_next_volume_limit": when next limit is reached (float) "money_amount": current money amount on card (float) "creation_tst": creation timestamp in epoch time (int) "creator_name": staff who created card (string) "last_cardpurchase_tst": timestamp when last cardpurchase occurred in epoch time (int) "last_staff": last staff who interacted with card (string) "last_tap_tst": last tap done with this card in epoch time (int) "expiration_tst": expiration timestamp set for card in epoch time (int) "currently_in_use_by_device": (string). Id of device which is currently using card (only applicable for online cards)
	JSON array "system_cards"	"card_id": id of card (int) "is_blocked": true/false (Boolean) "card_type": see table card types * (string) "card_name": name of card or staff who owns card (string) "card_creator": name of creator of card (string) "creation_tst": timestamp of creation of card in epoch time (int) "expiration_tst": expiration timestamp set for card in epoch time (int)

Card types *)

Type	*Description*

PRE	Pre pay card
BILL	Bill pay card
BILL POS	Simple bill pay card (used from certain POS interfaces)
ONLINE PRE	Online Pre pay card
ONLINE BILL	Online Bill pay card
ADMIN	Admin card
STAFF	Staff card
INSTALL	Install card
MANAGER	Manager card
CLEAN	Cleaning card
PERMANENT_BILL	Permanent login (e.g. for hotel lobbies)
EXT_CARD	External card – non PMB card
REMOTE_LOGIN	Remote login (for app usage)
OPEN_POUR	Open pour login
QR-CODE	Login via QR-code (instead of card)

3.21 Get transactions

Usage

Get all transactions from the database in a given timeframe

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/transactions
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "start_time": "YYYY-MM-DD hh:mm:ss" Timestamp (string) "end_time": "YYYY-MM-DD hh:mm:ss" Timestamp (string)

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload		„uuid": uuid of server (string)
	JSON array "taptransactions"	"card_id": id of card (int) "card_type": see table card types * (string) "name_customer" name of customer (string) "tst_start": timestamp of when the pour started in epoch time (int) "tst_stop": timestamp of when the pour stopped in epoch time (int) "plu": number of plu (product number) tapped (int) "money_amount": money amount tapped (float) "volume_amount": volume amount tapped (float) "servings": servings consumed with this tap (float) "external_id": used by some external interfaces "customer_hash": used for statistical purposes "device_id": id of device where tap occurred "item_id": when portioned mode the id of the item which was tapped/selected

3.22 Get cardpurchases

Usage

Get all cardpurchases from the database in a given timeframe

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/cardpurchases
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "start_time": "YYYY-MM-DD hh:mm:ss" Timestamp (string) "end_time": "YYYY-MM-DD hh:mm:ss" Timestamp (string)

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload		„uuid": uuid of server (string)
	JSON array "cardpurchases"	"card_id": id of card (int) "card_type": see table card types * (string) "tst_purchase" timestamp of cardpurchase action in epoch time (int) "tst_expiry": expiration timestamp when card was created/reactivated, etc. in epoch time (int) "money_amount": money amount tapped (float) "servings_limit": set servings limit (float) "card_action": card action – see table for details (int) "name_staff": name of staff which performed action (string) "name_customer": name of customer card for which action was performed (string) "customer_hash": used for statistical purposes

Card purchase types *)

Action		*Type*	*Details*
Pre pay	Create card	11	amount=0
	Reactivate card	12	amount=0
	Increase card value	13	amount=value to be added
	Balance card	19	amount=0, servings limit=0
Online Pre pay	Create card	111	amount=0
	Reactivate card	112	amount=0
	Increase card value	113	amount=value to be added
	Balance card	119	amount=0, servings limit=0
Bill pay	Create card	21	amount=0
	Reactivate card	22	amount=0
	Balance card	29	amount=0, servings limit=0
Online Pre pay	Create card	121	amount=0
	Reactivate card	122	amount=0
	Balance card	129	amount=0, servings limit=0
Create Admin card		31	amount=0, servings limit=0, expiration timestamp=0
Create Staff card		41	amount=0, servings limit=0, expiration timestamp=0

Create Cleaning card	51	amount=0, servings limit=0, expiration timestamp=0
Create Install card	61	amount=0, servings limit=0, expiration timestamp=0
Create Manager card	71	amount=0, servings limit=0, expiration timestamp=0
Erase Pre/Bill pay card	99	amount=0, servings limit=0
Erase system card	100	amount=0, servings limit=0
Erase Online Pre/Bill pay card	199	amount=0, servings limit=0
External card create	191	amount=0
QR Restalia/Codisys create	511	

3.23 Get serverconfig

Usage

Get all parameters from configuration table in config.sqlite of server

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/serverconfig
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	JSON array "configmap"	Returns all key/values that are stored in the configuration database

3.24 Set serverconfig

Usage

Set parameters for configuration table in config.sqlite of server

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/setserverconfig
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client
JSON payload (parameters)	"access_token": access token (string) for CESK WMO PLC API "activate_general_network_broadcast": 1/0 (Boolean)- Also send a broadcast at address "255.255.255.255" from TTG server to clients instead of network specific broadcast (e.g. 192.168.0.255) "allow_batch_balance" 1/0 (Boolean) – If activated you can batch balance cards on a charger. "allow_card_balance_without_login": 1/0 (Boolean) – card balancing can be done on a charger without the need to login to charger. Caution: Use only in locations where chargers are in a save location where no customer can access it! "allow_create_ticket": 1/0 (Boolean) – for PMB magic V2. Allow the "create ticket" operation on a charger or not "allow_gold_cards": 1/0 (Boolean) – for PMB magic V2. Activate gold card program "allow_online_cards": 1/0 (Boolean) – online cards can be created or not "allow_open_tickets_list": 1/0 (Boolean) – See the open ticket list on a charger or not "allow_pour_without_customer_card": 1/0 (Boolean) – Allow for online cards that you can pour without the need to leave the card in a card reader "allow_standard_pre_and_billpay_cards": 1/0 (Boolean) – for PMB magic V2. Allow the creation of default REDL desfire customer cards (as backup or for VIP customers, etc.) "allow_tap_for_crewmembers": 1/0 (Boolean) – costa interface. Pouring with crewmember cards possible or not "allow_unlimited_customer_cards": 1/0 (Boolean) – customer cards without expiration date can be created or not "api_key": (string). API key for authentication. Can be used for hotel interfaces: Web POS, PMB Online, WeWork, ibelsa.rooms "authorization_amount": For USA tech terminal (card reader type). (int) Must be positive "auto_logout": 1/0 (Boolean). Auto logout activated or not "bill_pay": 1/0 (Boolean) – for TiPOS remote interface. Allow bill pay cards in this interface or not "card_encpass": card encryption phrase (string). Must be alphanumeric and a maximum of 16 chars "card_uid_hex": 1/0 (Boolean) – for TiPOS remote interface. Card UID is in hex or not "card_uid_inverted": 1/0 (Boolean) for TiPOS remote interface. Card uid comes inverted or normal "cardreader_type": 1 – RFID Card reader (default) 2 – Magnetic Card reader 3 – Vendotek Terminal (serial) 4 – USA Technologies Terminal (Credit card terminal) –

	can only be used in the US 5 – CCV Terminal "company_code": (string). For CESK WMO PLC API integration "confirmation_needed": 1/0 (Boolean) – if activated and client is in portion mode each chosen portion needs to be confirmed before tap can happen and money is deducted "currency_unit_prefix": Currency prefix sign/symbol (string). Default is $. "currency_unit_suffix": Currency suffix sign/symbol (string). Default is NULL/empty "customfield_name": name of customfield for ibelsa.rooms integration "customfield_setvalue1": value field for ibelsa.rooms integration "customfield_setvalue2": value field for ibelsa.rooms integration "date_format_string": currently not in use and therefore cannot be set "daystart_offset": start of day offset (for certain operations in client, etc.). must be between -24 and +24 (int) – default is "2" "decimal_sep": decimal separator (string). Must be "." Or ",". Default is "." "default_bg_image": ID of default background image (int). Must match id of background images table in config database "default_fl_image": ID of default filllevelbar image (int). Must match id of filllevelbar images table in config database "default_language": language for clients/chargers (string). Currently supported: "en" = English, "de" = german, "pt" = Portuguese "disable_logout_button": 1/0 (Boolean) – disable logout button when auto logout is active "disable_logout_timer_for_system_cards": 1/0 (Boolean) – disable the logout timer for system cards (needed for NMO/MID certification) "domain": name of domain/server (string). Must be alphanumeric or underscore "domain_admin_user": username for administrator account "domain_admin_pass": password for administrator account "domain_client_user": username for client account "domain_client_pass": password for client account "driving_licence_scan_enabled": 1/0 (Boolean) – driving licence scanning activated for card creation process "driving_licence_statistics_enabled": 1/0 (Boolean) "driving_licence_statistics_enabled": 1/0 (Boolean) – generate customer_hash for every transaction/card purchase and store it in data_raw database. Can be used by other external tools for statistical purposes "dsa_limit_most_wanted_daily": not in use anymore "dsa_limit_most_wanted_weekly": not in use anymore "dsa_tapnumlabel_advertising": not in use anymore "dsa_tapnumlabel_most_wanted": not in use anymore "dsa_xml_cache_timeout" not in use anymore "enable_TTGServer_customer_card_reporting": 1/0 (Boolean) – for hotel interfaces: Omnivore, WebPOS, WeWork, Costa, PMB magic V2, MGM-Givex, CESK PLC API. If activated customer card reportings are still sent to TTG server and not only to hotel interface "enable_TTGServer_system_card_reporting": 1/0 (Boolean) – for hotel interfaces: Omnivore, WebPOS, WeWork, Costa, PMB magic V2, MGM-Givex, CESK PLC API. If activated system card reportings are still sent to TTG server and not only to hotel interface "enable_client_charger_switch": 1/0 (Boolean). Enables the possibility to switch a client to a charger as a "backup" charger "exist_changes_unexported": Needed for TTG internal purposes. Do not change! "exist_changes_unnotified": Needed for internal TTG processes. Do

	not change! "external_link_enabled": 1/0 (Boolean) - external link for TTG server main page activated "fillup_enabled": 1/0 (Boolean) – fill up for staff card enabled or disabled "fillup_enabled_manager": 1/0 (Boolean) – fill up for manager card enabled or disabled "get_product_name_from_tipos": 1/0 (Boolean) – for hotel interface TiPOS remote. Get the product names due to PLU number from TiPOS instead of TTG server "gold_cards_program_name": name for gold card program (string) – for PMB magic V2 "hide_keg_level" 1/0 (Boolean) – hide the keg level bar on the client or not "hotel_authentication": 1 – API-key, 2 – JWT. For certain hotel interfaces where you can choose the type of authentication. (WebPOS, PMB Online, WeWork) "hotel_interface": 0 – No hotel interface 1 – Resco SMS 2 – unTill TPAPI 3 – micros RES POS API 4 – TiPOS Remote 5 – weezevent API 6 – MSC Fidelio 7 – Blueticket BeerWallService 8 – TiPOS QR-Code 9 – Omnivore 10 – Web POS API 11 – PMB Online API 12 – WeWork 13 – Costa 14 – PMB Magic V2 15 – MGM-GiveX 16 – ibelsa.rooms 17 – CESK PLC API "hotelserver_backup_url": Backup url for certain hotel interfaces (WebPOS, PMB Online, PMB Magic V2, MGM-Givex "hotelserver_clientkey": clientkey for interface ibelsa.rooms "hotelserver_password": password for hotel interface authentication "hotelserver_systemkey": systemkey for interface ibelsa.rooms "hotelserver_url": URL for hotel interface to connect to "hotelserver_username": username for authentication for hotel interface "interface_admin_id_1": for costa interface. ID for admin card "interface_admin_id_2": for costa interface. ID for admin card "interface_admin_id_3": for costa interface, ID for admin card "interface_admin_id_4": for costa interface, ID for admin card "interface_admin_id_5": for costa interface, ID for admin card "interface_cleaning_id_1": for costa interface, ID for cleaning card "interface_cleaning_id_2": for costa interface, ID for cleaning card "interface_cleaning_id_3": for costa interface, ID for cleaning card "interface_cleaning_id_4": for costa interface, ID for cleaning card "interface_cleaning_id_5": for costa interface, ID for cleaning card "keg_level_illustration": for keg management page. Cannot be set via API "keg_level_sorting": for keg management page. Cannot be set via API "legal_drinking_age": legal drinking age setting (int) "license_key": licence key of TTG server. Cannot be changed "local_backup_dirpath": path for backup saves "local_backup_interval": interval in minutes for backups "local_backup_keep_num": number of backups to keep "local_user_name": username for local user "local_user_pass": password for local user

	"logging_enabled": 1/0 (Boolean). Client logging enabled or disabled "mail_recipient": email address (string) for email notifications where mails should be sent to "notif_1_subject": subject for notification 1 "notif_1_text": text for notification 1 "notif_1_threshold": threshold value for notification 1 "notif_2_subject": subject for notification 2 "notif_2_text": text for notification 2 "notif_2_threshold": threshold value for notification 2 "notif_3_subject": subject for notification 3 "notif_3_text": text for notification 3 "ntp_server": url of ntp server "only_one_article_per_session": 1/0 (Boolean) – for TiPOS remote interface. Allow only 1 tap per session "pressure_monitoring": 1/0 (Boolean). If enabled a central pressure monitoring for all pressure gas is active "price_tag_color": 1 = white, 2 = yellow (int) "responsibility_text_1": text for responsibility limit message on client (string) "responsibility_text_2": deprecated. Currently not in use "servings_per_patron": servings per patron setting (int) e.g. 200 for 2.00 "show_percent_portion_mode": 1/0 (Boolean). Show percent instead of volume in portion mode when tapping "show_poured_amount" 1/0 (Boolean). Show poured money amount on logout if enabled "show_poured_volume" 1/0 (Boolean). Show poured volume on logout if enabled "show_poured_volume_per_product" 1/0 (Boolean). Show poured volume per product on logout if enabled "show_price_label" 1/0 (Boolean). Show price tag in logged out state (no card present) for product prices (only in standard mode, not in portion mode) "show_unit_price" 1/0 (Boolean). Show price per unit when logged in (in standard mode, not in portion mode) "smtp_password": SMTP password for SMTP server (for notifications needed) "smtp_port": SMTP port to connect to for SMTP server (for notifications) "smtp_server": SMTP server to connect to (for notifications) "smtp_user": SMTP user for SMTP server (for notifications) "staff_code": (int), for TiPOS remote interface "statistics_server": (string) – currently not in use "tap_time_activate_limitation": 1/0 (Boolean) – tap time limitation enabled or disabled "tap_time_limit_end": End allowed time in minutes (int) "tap_time_limit_start": Start allowed time in minutes (int) "tap_time_limitation_day_specific": currently not in use "tap_time_text_1": tap time limitation text (string) "tap_time_text_2": currently not in use, deprecated "time_zone": timezone setting for NTP server (provide time zone in proper format) "timeout_setting": time in seconds until timeout (int) for MGM-Givex interface "use_notif_mail_1": 1/0 (Boolean). Enable/Disable notification 1 "use_notif_mail_2": 1/0 (Boolean). Enable/Disable notification 2 "use_notif_mail_3": 1/0 (Boolean). Enable/Disable notification 3 "use_notif_ttg_1": for future use – not in use currently "use_notif_ttg_2": for future use – not in use currently "use_notif_ttg_3": for future use – not in use currently "uuid": cannot be changed by API or user "validity_period": validity period in minutes (int). for interface TiPOS QR-Code

	"version": config db version – cannot be changed "volume_base_unit": volume base unit (string). Default is oz "volume_base_unit_dp": number of decimal points for volume (int). Default is 1 "waiter_id": id of waiter (int). Used in hotel interfaces: unTill, MICROS, ibelsa.rooms "week_date_conventions": (string). Possible values: "US", "ISO"

Response from server
HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	Returns success or error for each key/value pair

3.25 Get HappyHours

Usage

Get all configured happy hours from the server

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/gethappyhours
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload		„uuid": uuid of server (string)
	JSON array "happyhours"	"group_id": group id of happy hour (string) "happy_id": unique id of happy hour (int) "happy_name": name of happy hour (string) "week_day_start": if weekday specific > 0 either 0 (0= no weekday, 1 = Monday, 2 = Tuesday, etc.) "start_date": if date specific date in "YYYY-MM-DD" format. "" if emtpy "start_time": time of day in minutes "week_day_end": if weekday specific > 0 either 0 (0= no weekday, 1 = Monday, 2 = Tuesday, etc.) "end_date": if date specific date in "YYYY-MM-DD" format. "" if emtpy "end_time": time of day in minutes "keg_level": if keg level specific keg level in % when happy hour triggers "keg_age": if keg age specific, keg age in days "price_level": price level of happyhour "affected_plus": affected plus of happyhour. If empty ("") all plus affected "affected_product_types": affected product type of happy hour. If empty ("") all product types affected

3.26 Change Happy Hour

Usage

Change one or more happy hours

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/changehappyhour
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>		TTG server IP address
<HTTP port>		TTG server HTTP port
<token>		Authorization token
JSON payload		"id": id of the API client
	JSON array "happyhours"	"happy_id": unique id of happy hour (int) "happy_name": name of happy hour (string) "week_day_start": if weekday specific > 0 either 0 (0= no weekday, 1 = Monday, 2 = Tuesday, etc.) "start_date": if date specific date in "YYYY-MM-DD" format. "" if emtpy "start_time": time of day in minutes "week_day_end": if weekday specific > 0 either 0 (0= no weekday, 1 = Monday, 2 = Tuesday, etc.) "end_date": if date specific date in "YYYY-MM-DD" format. "" if emtpy "end_time": time of day in minutes "keg_level": if keg level specific keg level in % when happy hour triggers "keg_age": if keg age specific, keg age in days "price_level": price level of happyhour "affected_plus": affected plus of happyhour. If empty ("") all plus affected "affected_product_types": affected product type of happy hour. If empty ("") all product types affected

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.27 Delete Happy Hour

Usage

Delete single Happy Hour or whole group of happy hours (if weekday specific)

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/deletehappyhour
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "delete_id": happy hour id (or group id) to delete (int) "isgroupid": 0/1 (true/false) determine if provided id is a group_id (true) or happy_id (false)

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.28 Add Happy Hour

Usage

Add one or more happy hours

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/addhappyhour
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>		TTG server IP address
<HTTP port>		TTG server HTTP port
<token>		Authorization token
JSON payload		"id": id of the API client
	JSON array "happyhours"	"happy_id": unique id of happy hour (int) "group_id": group id of happy hour (string) "happy_name": name of happy hour (string) "week_day_start": if weekday specific > 0 either 0 (0= no weekday, 1 = Monday, 2 = Tuesday, etc.) "start_date": if date specific date in "YYYY-MM-DD" format. "" if emtpy "start_time": time of day in minutes "week_day_end": if weekday specific > 0 either 0 (0= no weekday, 1 = Monday, 2 = Tuesday, etc.) "end_date": if date specific date in "YYYY-MM-DD" format. "" if emtpy "end_time": time of day in minutes "keg_level": if keg level specific keg level in % when happy hour triggers "keg_age": if keg age specific, keg age in days "price_level": price level of happyhour "affected_plus": affected plus of happyhour. If empty ("") all plus affected "affected_product_types": affected product type of happy hour. If empty ("") all product types affected

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

3.29 Remote Login request

Attention: client control API needs to be activated (Enable HTTP listener in server configuration)

Usage

Perform a remote login (without card) at a specific client with a specific pre-selected portion. If executed correctly the TTG client will login and preselect the portion and waits for confirmation of payment.

Request from client

HTTP POST request to *<server IP address>*:*<HTTP port>*:

```
POST /m2m/api/remoteLogin
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "cardId": id of customer/card who wants to login (string) "valid": 0/1 - is valid for logging in to TTG system (Boolean) "name": name of customer (string) "balance": already consumed money amount (int) "credit": remaining credit for customer (-1 if infinite = billpay) (int) "isAllowedToPourAlcohol": 0/1 – can pour alcoholic beverages or not (ABV > 0 = alcohol) "type": customer account type (int) 1=customer, 2=cleaning, 3=staff, 4=admin, 5=guest "deviceID": id of device for which login is requested (int) "line": line number of device which is affected (1-4) (int) "volumeConsumed": already consumed volume of customer (int) "volumeLimit": requested portion size in units with 3 decimal points (int) "volumeLimitSession": requested portion size in units with 3 decimal points (int) – must be same like volumeLimit "transactionId": unique transaction id (string) for external system (e.g. for payment, etc.)

Response from server

HTTP response to *<client IP address>*:*<HTTP port>*:

```
HTTP 200 – OK
```

JSON payload	"cardId": id of customer/card which requested login "transactionId": unique transaction id (string) for external system (e.g. for payment, etc.) "deviceID": id of device for which login is requested (int) "line": line number of device which is affected (1-4) (int) "requestedPortion": requested portion name "volume": requested portion size in units with 3 decimal points (int) "volumeUnit": unit for portion (e.g. ml) (string) "moneyAmount": price for requested portion in cents (int) "productName": name of requested product (string)

	"productPLU": PLU (TTG internal product number) of product (int) "product-type-number": product type number **) (int) "product-type-name": product type description (string) "description": [productName] – [portionName] (string) "error": 0 = no error, otherwise see errorlist *) (int) "errorDescriptionAPI": if error > 0 and error appeared in TTG API (string) "errorCodeClient": if error > 0 and error happened in TTG client (int) "errorDescriptionClient": if error > 0 and error happened in TTG client (string)

*)

Error-Code	*Details*
0	No error everything is ok
100	Invalid "type" specified in request
101	Invalid line specified in request (line not configured on client or > 4)
102	Invalid requested portion size (not available on client)
103	Currently someone is logged in at client and logout is not possible
200	Workflow error on client (internal error)
201	Portion could not be selected on client (internal error)
202	Insufficient funds for selected portion
203	Requested product was alcoholic but customer cannot pour alcohol (isAllowedToPourAlcohol set to 0)
204	Not enough volume in beer keg left – portion cannot be poured
205	Update value error
206	Servings limit reached
300	Invalid cardId provided
301	Invalid transactionId provided
400	Payment could not be completed
4000	Invalid JSON request (missing parameter or parameter with wrong values – see error description)
4001	Client not found: No TTG client found with given deviceID which is connected to the TTG server
4002	Requested portion size not configured/available on given device on that line
4003	Requested line not found for given deviceID
5000	Unexpected error in API – details see error description
5004	TTG/tap client timeout. Requested client cannot be reached via network (either client is down or network connection broken/offline)

**)

Product-Type	*Product-Type-Name*
0	Unknown/Not set
1	Beer
2	Wine
3	Cocktail
4	Non-Alcoholic
5	Games
6	Food
7	Cider

3.30 Remote Login Purchase request

Attention: client control API needs to be activated (Enable HTTP listener in server configuration)

Usage

After remoteLogin request. With response of remoteLogin request all needed parameters are returned for handling payment. If payment is ok you can call remoteLoginPurchase to finish login procedure so customer can tap selected portion on client

Request from client

HTTP POST request to *<server IP address>*:*<HTTP port>*:

```
POST /m2m/api/remoteLoginPurchase
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "cardId": id of customer/card who wants to login (string) "transactionId": unique transaction id (string) for external system (e.g. for payment, etc.) "paymentId": unique payment id (string) for external system which is provided by external payment system "error": 0 if payment is completed without errors. If error > 0 client will abort login

Response from server

HTTP response to *<client IP address>*:*<HTTP port>*:

```
HTTP 200 – OK
```

JSON payload	"cardId": id of customer/card which requested login "transactionId": unique transaction id (string) for external system (e.g. for payment, etc.) "paymentId": unique payment id (string) for external system which is provided by external payment system "error": 0 = no error, otherwise see errorlist *) (int) "errorDescriptionAPI": if error > 0 and error appeared in TTG API (string) "errorCodeClient": if error > 0 and error happened in TTG client (int) "errorDescriptionClient": if error > 0 and error happened in TTG client (string)

*)

Error-Code	*Details*
0	No error everything is ok
100	Invalid "type" specified in request
101	Invalid line specified in request (line not configured on client or > 4)
102	Invalid requested portion size (not available on client)
103	Currently someone is logged in at client and logout is not possible
200	Workflow error on client (internal error)
201	Portion could not be selected on client (internal error)
202	Insufficient funds for selected portion
203	Requested product was alcoholic but customer cannot pour alcohol

	(isAllowedToPourAlcohol set to 0)
204	Not enough volume in beer keg left – portion cannot be poured
205	Update value error
206	Servings limit reached
300	Invalid cardId provided
301	Invalid transactionId provided
400	Payment could not be completed
4000	Invalid JSON request (missing parameter or parameter with wrong values – see error description)
4001	Client not found: No TTG client found with given deviceID which is connected to the TTG server
4002	Requested portion size not configured/available on given device on that line
4003	Requested line not found for given deviceID
5000	Unexpected error in API – details see error description
5004	TTG/tap client timeout. Requested client cannot be reached via network (either client is down or network connection broken/offline)

3.31 Get Portions for Device and Line request

Usage

Get a list of portions (items) which are configured for a specific line on a specific device

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/getPortionsForDeviceAndLine
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "deviceID": id of the device where you want to know the items "line": line number of the target device (1-4)

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	"uuid": uuid of the server (string) "deviceId": id of the client for the asked portions/items "line": line number of where the portions/items are configured	
	JSON array "portions"	"product-name": name of the product configured "product-type-number": product type of the product as number "product-type-name": product type of the product as name "portion-name": name of the portion for this item "portion-volume": volume of the portion "portion-volume-dp": decimal point setting for the volume "portion-volume-unit": unit used for portion volume "portion-price": price of the portion/item in cents (e.g. 200 = 2.00) "portion-price-currency-prefix": currency prefix string "portion-price-currency-suffix": currency suffix string

3.32 Restart client application request

Attention: client control API needs to be activated (Enable HTTP listener in server configuration)

Usage

Restarts the TTG application on a specific client

Request from client

HTTP POST request to *<server IP address>*:*<HTTP port>*:

```
POST /m2m/api/restartClientApplication
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "deviceID": id of the device

Response from server

HTTP response to *<client IP address>*:*<HTTP port>*:

```
HTTP 200 – OK
```

JSON payload	"deviceId": id of the client "error": 0 if no error "errorDescriptionAPI": If a error happened in the TTG server API "errorCodeClient": If the error happened on the client side "errorDescriptionClient": If the error happened on the client side

3.33 Get device count

Usage

Get the number of devices currently connected to the TTG server

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/getdevicecount
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"old_count": current number of keg levels (int)

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	"devicecount": current count of devices connected to TTG server

3.34 Get device status request

Attention: client control API needs to be activated (Enable HTTP listener in server configuration)

Usage

Retrieve detailed status information from a specific device

Request from client

HTTP POST request to *<server IP address>*:*<HTTP port>*:

```
POST /m2m/api/getdevicestatus
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"deviceID": id of the device

Response from server

HTTP response to *<client IP address>*:*<HTTP port>*:

```
HTTP 200 – OK
```

JSON payload	"deviceId": id of the client "cpu_usage": current cpu usage of the client "current_date_time": Current date/time of the client "data_disk_free": free disk space "data_disk_used": used disk space "mem_available": Free memory available "mem_used": memory in use "swap_available": swap available "swap_available ": swap in use "system_disk_free": free system disk space "system_disk_free": used system disk space "ttg_server_auth_token": TTG server auth token used "ttg_server_comm_state": Communication state with TTG server (1=connected, 0 = connection broken) "ttg_server_domain": TTG server domain name which the client is connected to "ttg_server_ip": IP of the TTG server which the client is connected to "ttg_server_port": Port of the TTG server which the client is connected to "uptime": current uptime of the client "error": 0 if no error "errorDescriptionAPI": If error happened on TTG Server control side "errorCodeClient": If error happened on client side "errorDescriptionClient": If error happened on client side

3.35 Set Backlight Brightness request

Attention: client control API needs to be activated (Enable HTTP listener in server configuration)

Usage

Set the backlight brightness of a specific device. Info: The backlight brightness will be restored to 100% after a restart of the client

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/setBacklightBrightness
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "deviceID": id of the device "brightness": desired brightness in %

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	"deviceId": id of the client "brightness": new brightness of the client in % "error": 0 if no error "errorDescriptionAPI": If error happened on TTG Server control side "errorCodeClient": If error happened on client side "errorDescriptionClient": If error happened on client side

3.36 Get Server basic info request

Usage

Retrieve server version, build number, config version, current hotel interface used from a TTG server

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/getServerBasicInfo
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	"uuid": uuid of the server "TTG_server_version": current server software version "TTG_server_build": current server software build number "TTG_server_config_version": current server database version "hotel_interface_enum": current hotel interface in use (enum) "hotel_interface_text": Current hotel interface (text)

Hotel-Interfaces

Enum	*Text (Description)*
0	Standard mode – no integration
1	Resco SMS
2	until TPAPI
3	Micros RES POS API
4	TiPOS Remote
5	Weezevent API
6	MSC Fidelio
7	Blueticket BeerWallService
8	TiPOS QR-Code
9	PMB Magic v1 (Omnivore)
10	Web POS API
11	PMB Online API
12	WeWork
13	Costa PMB API
14	PMB Magic v2
15	Givex
16	Ibelsa.rooms
17	CESK WMO PLC API

3.37 Get driving licence data

Usage

Get all driving licence data from the database in a given timeframe

Request from client

HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/getDrivingLicenceData
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "start_time": "YYYY-MM-DD hh:mm:ss" Timestamp (string) "end_time": "YYYY-MM-DD hh:mm:ss" Timestamp (string)

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload		„uuid": uuid of server (string)
	JSON array "driving-licence-data"	"id": internal database id of the driving license "device_id": from which charger (device) the license data was sent "card_id" associated cardId with the license "dob": Date of birth stored on the driving license "sex": Sex stored on the driving license "zip_code": zip_code on the driving license "iin": IIN number of the driving license "timestamp_epoch": Timestamp when the driving license was scanned in epoch time "timestamp_hr": Timestamp when the driving license was scanned in human readable format [YYYY-mm-dd HH:MM:SS]

3.38 Get Timestamp Config Export

Get the current timestamp when the config export database was last created/updated

Request from client
HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/gettimestampconfigexport
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client

Response from server
HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	"uuid": uuid of the server "tst_cfg_export_last_update_epoch": Timestamp in epoch time when config export db was recently created/updated "tst_cfg_export_last_update_hr": Timestamp in human readable format time when config export db was recently created/updated [YYYY-mm-dd HH:MM:SS]

3.39 Send cardpurchase data

Usage

Send a cardpurchase to the server (like a TTG charger)

Request from client

HTTP POST request to *<server IP address>*:*<HTTP port>*:

```
POST /m2m/api/cardpurchase
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address> <HTTP port> <token>	TTG server IP address TTG server HTTP port Authorization token	
JSON payload		"id": id of the API client "locationID": ID of the location which sent the request (int) "deviceID": ID of the device which sent the request (int)
	JSON array "cardpurchases"	"card_id": UID of the card in decimal format (int) "creationTimestamp" Timestamp when card action was performed (card created) [YYYY-mm-dd HH:MM:SS] (string) "expirationTimestamp": (string) can be empty or in [YYYY-mm-dd HH:MM:SS] format when card should expire and cannot be used anymore "cardPurchaseType": card purchase type (int) – see table for further information "amount": money amount in cents that is loaded onto the card (int) "servingsLimit": number of servings that are possible before card needs reactivation (int) – set to 0 to disable "cardName": name of the card/customer "staffName": name of the staff/system that created the card "deposit": money amount in cents for deposit of the card (used for self-serve charger only currently) "alcohol_allowed": 0 to disable 1 to enable, only if enabled customer is allowed to pour alcoholic drinks

Card purchase types *)

Action		*Type*	*Details*
Pre pay	Create card	11	Not possible with the API (offline card)
	Reactivate card	12	Not possible with the API (offline card)
	Increase card value	13	Not possible with the API (offline card)
	Balance card	19	Not possible with the API (offline card)
Online Pre pay	Create card	111	amount=0
	Reactivate card	112	amount=0
	Increase card value	113	amount=value to be added
	Balance card	119	amount=0, servings limit=0
Bill pay	Create card	21	Not possible with the API (offline card)
	Reactivate card	22	Not possible with the API (offline card)
	Balance card	29	Not possible with the API (offline card)
Online Pre pay	Create card	121	amount=0
	Reactivate card	122	amount=0
	Balance card	129	amount=0, servings limit=0

Create Admin card	31	Not possible with the API (offline card)
Create Staff card	41	Not possible with the API (offline card)
Create Cleaning card	51	Not possible with the API (offline card)
Create Install card	61	Not possible with the API (offline card)
Create Manager card	71	Not possible with the API (offline card)
Erase Pre/Bill pay card	99	Not possible with the API (offline card)
Erase system card	100	Not possible with the API (offline card)
Erase Online Pre/Bill pay card	199	amount=0, servings limit=0
External card create	191	Not possible with the API (offline card)
QR Restalia/Codisys create	511	Not possible with the API

Response from server
HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	„uuid”: uuid of server (string) “valid”: true/false (bool) – true if sent cardpurchase was valid (no errors) – false when there is something missing or in wrong format “records_inserted”: (int) number of records inserted into database “records_skipped”: (int) number of records skipped because they already exist in database “records_skipped_str”: (string) information which record was skipped with which timestamp (in epoch time) “records_malformatted”: (int): number of cardpurchases malformatted if valid = false “infostring”: (string) – if there are malformatted cardpurchases here is further explanation what is wrong (missing parameter, wrong format, etc.)

3.40 cardinfo

Get cardinfo for specific card

Request from client
HTTP POST request to <*server IP address*>:<*HTTP port*>:

```
POST /m2m/api/cardinfo
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address>	TTG server IP address
<HTTP port>	TTG server HTTP port
<token>	Authorization token
JSON payload	"id": id of the API client "locationID": (string) id of the location "deviceID": (int), id of client which sends the request. 0 can be used if no login with card is intended "login" true/false. (bool). If true card will be locked until transaction is transmitted from client with this device id "cardID": uid of the card for which you want to have the cardinfo (int) in decimal format

Response from server
HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	"uuid": uuid of the server "cardID": (int) uid of the card in decimal format "name": customer/card name (string) "balance": already consumed money on this card in cents (int) "credit": (int) in cents. 0 if billpay card. Must be > 0 if it is a prepay card (remaining credit) "type": Type of card (int) *see table card types (Type(int)) "type_string": Type of card as string description (string) *see table card types (Type(string)) "consumedServings": Already consumed servings (int) "servingsLimit": Servings limit (int) "creationTimestamp": Timestamp when card action was performed (card created) [YYYY-mm-dd HH:MM:SS] (string) "lastTimestamp": Timestamp when last card action was performed [YYYY-mm-dd HH:MM:SS] (string) "expirationTimestamp": Timestamp when card expires, empty when no expiration set [YYYY-mm-dd HH:MM:SS] (string) "isBlocked": 0 (not blocked) or 1 (blocked) (int) "blockedTimestamp": Timestamp when card was blocked, empty when not blocked [YYYY-mm-dd HH:MM:SS] (string) "blockedComment": Comment/additional info on blocked card, empty when not blocked (string) "staffCreationName": staff name which created the card (string)

Card types *)

Type (int)	*Type (string)*	*Description*
1	PRE	Pre pay card
2	BILL	Bill pay card
21	BILL POS	Simple bill pay card (used from certain

		POS interfaces)
11	ONLINE PRE	Online Pre pay card
12	ONLINE BILL	Online Bill pay card
3	ADMIN	Admin card
4	STAFF	Staff card
6	INSTALL	Install card
7	MANAGER	Manager card
10	CLEAN	Cleaning card
32	PERMANENT_BILL	Permanent login (e.g. for hotel lobbies)
19	EXT_CARD	External card – non PMB card
40	REMOTE_LOGIN	Remote login (for app usage)
33	OPEN_POUR	Open pour login
51	QR-CODE	Login via QR-code (instead of card)
61	CARD_TYPE_MDB_PAYMENT_PRE	MDB payment (e.g. card terminal) - prepay
62	CARD_TYPE_MDB_PAYMENT_BILL	MDB payment (e.g. card terminal) - billpay

3.41 Send transaction data

Usage

Send a transaction to the server (taptransaction like a TTG client)

Request from client

HTTP POST request to *<server IP address>*:*<HTTP port>*:

```
POST /m2m/api/transaction
Content-Type: application/json
Host: <server IP address>:<HTTP port>
Authorization:
JWT: HTTP-Header: Authorization=Bearer <Token>
```

<server IP address> <HTTP port> <token>	TTG server IP address TTG server HTTP port Authorization token	
JSON payload		"id": id of the API client "locationID": ID of the location which sent the request (string) "deviceID": ID of the device which sent the request (int)
	JSON array "transactions"	"card_id": UID of the card in decimal format (int) "card_type": type of card (int) – see table for further information "tst_start" Timestamp when card action was started (tap started) [YYYY-mm-dd HH:MM:SS] (string) "tst_stop": (string) Timestamp when card action was ended (tap stopped) [YYYY-mm-dd HH:MM:SS] (string) "plu": product number (int) – must exist in the server,), 0 to identify an empty transaction which releases the given card for usage on other clients "money_amount": money value in cents (int32), 0 to identify an empty transaction which releases the given card for usage on other clients "volume_amount": poured volume in units (int32), 0 to identify an empty transaction which releases the given card for usage on other clients "volume_amount_dp": number of decimal places used for volume (int) "servings": the poured servings (int32) e.g. 135 = 1.35 "external_id": card_id OR own customer identification (string), must not be empty "card_name": name of the customer/card (string), must not be empty "price_per_unit": price per unit in cents (int32) "item_id": id of the item (string), must match the item id in the TTG server

Card types *)

Type (int)	*Type (string)*	*Description*
1	PRE	Pre pay card
2	BILL	Bill pay card
21	BILL POS	Simple bill pay card (used from certain POS interfaces)
11	ONLINE PRE	Online Pre pay card
12	ONLINE BILL	Online Bill pay card
3	ADMIN	Admin card
4	STAFF	Staff card
6	INSTALL	Install card

7	MANAGER	Manager card
10	CLEAN	Cleaning card
32	PERMANENT_BILL	Permanent login (e.g. for hotel lobbies)
19	EXT_CARD	External card – non PMB card
40	REMOTE_LOGIN	Remote login (for app usage)
33	OPEN_POUR	Open pour login
51	QR-CODE	Login via QR-code (instead of card)
61	CARD_TYPE_MDB_PAYMENT_PRE	MDB payment (e.g. card terminal) - prepay
62	CARD_TYPE_MDB_PAYMENT_BILL	MDB payment (e.g. card terminal) - billpay

The red marked entries make no sense to use via the TTG server control API as these are "offline" cards which communicate directly with the TTG server via the TTG tap client

Response from server

HTTP response to <*client IP address*>:<*HTTP port*>:

```
HTTP 200 – OK
```

JSON payload	„uuid": uuid of server (string) "valid": true/false (bool) – true if sent cardpurchase was valid (no errors) – false when there is something missing or in wrong format "records_inserted": (int) number of records inserted into database "records_skipped": (int) number of records skipped because they already exist in database "records_skipped_str": (string) information which record was skipped with which timestamp (in epoch time) "records_malformatted": (int): number of cardpurchases malformatted if valid = false "infostring": (string) – if there are malformatted cardpurchases here is further explanation what is wrong (missing parameter, wrong format, etc.)