KOLLYgram 15

Web API Documentation

BurningBox SA for Garage KOLLY SA

Content table

Revisions

Date	Ref.	Description
01.04.2015	QO	Initial document
16.04.2015	QO	Several fields updated
29.04.2015	QO	New API URL
05.05.2015	QO	Added Wash model and WashesController
13.05.2015	QO	Added Response status code + verify routing
08.06.2015	QO	UidLMU deleted, slaveId (byte) created, concerningMeasureId renamed into eventMeasureReceiptNumber (all in Measure)
19.06.2015	QO	oldCustomerNumber of model Address can be used as an external customer ID
26.06.2015	QO	Added the timestamp properties
14.07.2015	QO	Added actions to retrieve models by the timestamp
20.07.2015	QO	Added the data types and new properties for Vehicle
23.07.2015	QO	Disable GET/PUT/DELETE in the UserAccessesController Added a function to retrieve all UserAccesses of a User and his children
31.07.2015	QO	Added the Measure quality description
04.08.2015	QO	Added the event codes table
05.08.2015	QO	Added properties into brackets in the models description. Added a section explaining the event codes.
30.10.2015	QO	Removed the Measure property “Changes” (unused) Added the EventCode model + controller Added the EventCodeIcon model + controller
02.11.2015	QO	Added ts_uid to the model EventCode
23.11.2015	QO	Added Rince model and controller
20.01.2016	QO	Added functions in the MeasuresController and EventsController, corrected timestamp behavior
21.06.2016	YB	Added Orders Module
02.11.2016	YB	Add CustomerId and Timestamp where needed
03.24.2017	YB + QO	Add startDelay and endDelay in Order model
13.09.2019	QO	Added IdentChips and IdentCustomers
15.09.2020	QO	Corrected chip numbers authorized length
21.12.2020	QO	Added AddressType in Address model
14.01.2021	WG	Added Unique information on fields
09.06.2022	QO	Added missing fields in Order and DeliveryPoint
18.06.2022	WG	Added new customer field on DeliveryPoint
27.09.2022	QO	Added new weighingType field in Measure
20.10.2022	QO	Added new LinkedMeasureId field in Order
26.10.2022	QO	Added new filter toBeInvoicedSeparately in /Measures/FromId

Business description

This first chapter will explain the business this API is used for.

KOLLYgram 15 is a system that is able to weight dynamically what vehicles are carrying. In order to achieve that, the vehicles are equipped with a technology that is able to figure out the weight of the product the vehicles has lifted and send it to a distant server that will keep track of all measures.

The customer that uses the KOLLYgram system possesses many containers. They can be filled with products or garbage. When the container is full, a truck comes to empty it.

The main goal of the KOLLYgram system is to keep a precise history of the activities around those containers and vehicles. To do that, an electronic chip is installed on each container so that the truck can identify it when lifting. At that moment, the truck is able to weight precisely the container. The difference of weight before and after the emptying give a measure of weight. After a while, those measures are sent automatically to a server that will store them. At any time, the customer can analyze the records and is able to manage efficiently his business.

At the end, the customer pays according to the number of measures and the modules he’s using.

Models

This chapter will list all objects that can be handled through this API.

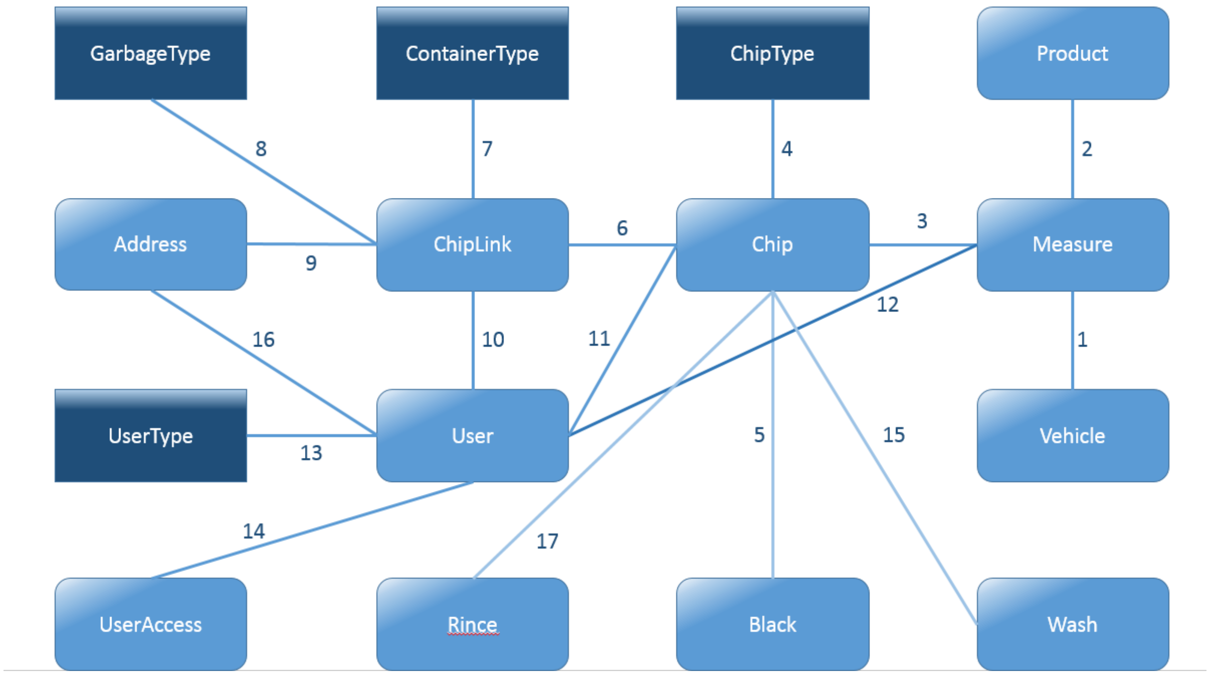


Figure 1 Relations between the models

The upper schema shows the relations between the models of the API.

A model is a class that represents a concrete object.

A model can be send and received through the controllers and manipulated by the client.

Here is a brief description of each link:

1. A Measure is generated by a Vehicle when lifting a container.
   (Measure.vehicleId = Vehicle.id)
2. A Measure is the weight of a Product at a precise time.
   (Measure.productNumber = Product.productNo)
3. A Measure is identified by a Chip that is installed on a container.
   (Measure.chipKey1/Measure.chipKey2 = Chip.chipNumber)
4. A Chip has a ChipType that specifies the technology of the Chip.
   (Chip.chipTypeId = ChipType.id)
5. A Chip can be blacklisted (for example if the owner of the container is a bad payer).
   (Black.chipId = Chip.id)
6. A ChipLink links a Chip with other models.
   (ChipLink.chipId = Chip.id)
7. A ChipLink possesses a ContainerType that specifies the size of a container.
   (ChipLink.containerTypeId = ContainerType.id)

8. A ChipLink has a GarbageType which informs the kind of garbage.
   (ChipLink.garbageTypeId = GarbageType.id)
9. A ChipLink possesses two Addresses.
   A ChipLink has a location Address which localizes the position of the container.
   A ChipLink has an invoice Address in order to send bills.
   (ChipLink.locationAddressId/ChipLink.invoiceAddressId = Address.id)
10. A ChipLink must have a User which is the customer.
    (ChipLink.customerId = User.id)
11. A Chip must have a User which is the customer that possesses the container.
    (Chip.customerId = User.id)
12. *(Optional)* A Measure may have a customer number. This number is entered by the vehicle driver directly. **Warning**: the customer number is not an ID.
    (Measure.customerNumber = User.customerNumber)
13. A User has a UserType which identifies the function of the User.
    (User.userType = UserType.id)
14. A User possesses UserAccesses (at least one) which specifies the rights of the User on the KOLLYgram system.
    (User.accesses.id = UserAccess.id)
15. A Chip can be wash listed. It means that the container carrying the chip needs to be washed.
    (Wash.chipId = Chip.id)
16. An Address belongs to a User. An Address can be used to localize a Chip and to bill Measures to the customer.
    (Address.customerId = User.id)
17. A Chip can be rinse listed. It means that the container carrying the chip needs to be rinsed.
    (Rince.chipId = Chip.id)

This second scheme describes all entities involved in the Orders module. Some of them are importable/exportable by the API, others are not. See details in the description below the scheme.

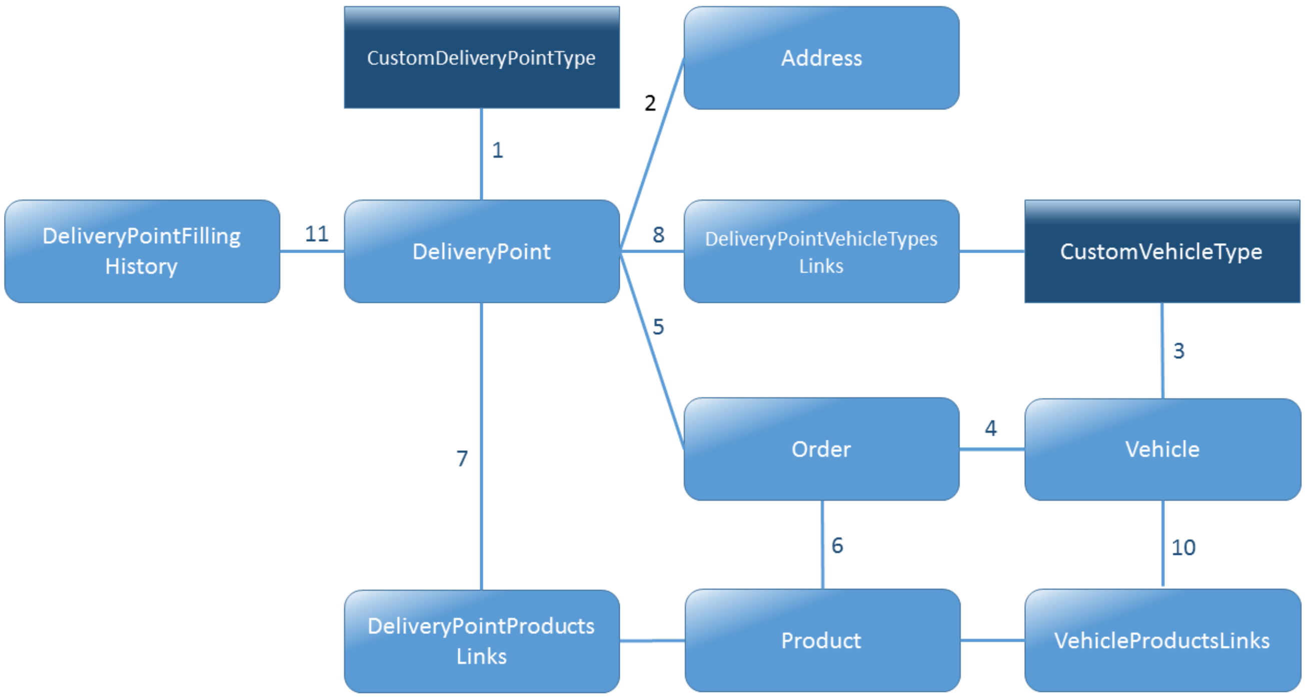


Figure 2 Relations between the orders module models

Here is a brief description of each link:

1. A DeliveryPoint describes a place for which it is possible to create an order for. It can be of different kinds: Delivery, Loading, Neutral, Delivery + loading. For each kind, the user can define custom types:
   (DeliveryPoint.deliveryPointTypeId = CustomDeliveryPointType.id)
2. A DeliveryPoint is linked to 2 addresses: one for the delivery, one for the billing.
   (DeliveryPoint.locationAddressId/ DeliveryPoint.invoiceAddressId = Address.id)
3. A Vehicle can be of different type. It is possible to define custom vehicle types, permitting to define a type name and if the vehicle is a trailer. Then each vehicle is linked to one of those vehicle types.
   (Vehicle.CustomVehicleTypeId = CustomVehicleType.Id)

An Order is a mission for a Vehicle to deliver a Product to a DeliveryPoint.

4. For each Order, there is a planned vehicle and the one that effectively did the job.
   (Order.plannedVehicleId / Order.deliveringVehicleId = Vehicle.id)
5. Each Order is linked to a DeliveryPoint.
   (Order.deliveryPointId = DeliveryPoint.id)
6. Each Order is linked to a Product.
   (Order.productNumber = Product. productNo)
7. A DeliveryPoint contains a list of supported Products that can be delivered to.
   (DeliveryPoint.supportedProductNumbers = Product.productNo, Product.productNo, …)

8. int contains a list of supported Vehicle types that can deliver to it.
   (DeliveryPoint.supportedVehicleTypeIds = CustomVehicleType.id, CustomVehicleType.id, …)
9. A DeliveryPoint contains a list of fillings that has been done in the past.
   (DeliveryPointFilling.deliveryPointId = DeliveryPoint.id)
10. A Vehicle contains a list of supported Products that the Vehicle can carry, and the quantity.
    (VehicleProduct.VehicleId = Vehicle.Id, VehicleProduct.productNumber = Product.No)

The following section will explain in further details the properties of each model.

For every model that have the Id field, this one must be 0 when adding. The unique Id value is only on existing models.

Measure

It represents a measure taken by a Vehicle **or an event**.

A Measure has the property “eventCode” set to 0. It’s an Event if “eventCode” is different than 0.

A Measure contains the following properties:

Property name	**Data type**	**Description**
id	Long, *unique*	ID of the Measure in the database
vehicleId	Long	ID of the Vehicle that took the Measure in the database
receivedOnServerAt	Date	Time the server received the Measure
netWeight	Decimal	Weight of the Measure
productNumber	Integer	Product number (is related to Product.ProductNo)
customerNumber	Integer	Customer number given by the carrier. This is not the ID of the customer in the database.
chipKey1	String (16)	The chip number of the first chip on the Vehicle.
dateAndTimeAtStart	Date	Time when the measure started
dateAndTimeAtEnd	Date	Time when the measure finished
kmAtStart	Decimal	The number of kilometers the Vehicle has driven at the beginning of the Measure
kmAtEnd	Decimal	The number of kilometers the Vehicle has driven at the end of the Measure
receiptNumber	Integer	Number written on the receipt
latitudeAtMeasureStart	Double	Location (latitude) at the beginning of the measure
longitudeAtMeasureStart	Double	Location (longitude) at the beginning of the measure
chipKey2	String (16)	The chip number of the second chip on the Vehicle
deleted	Boolean	True if the Measure has been deleted, False otherwise
gross	Decimal	Gross weight
tare	Decimal	Tare weight
longitudeAtMeasureEnd	Double	Location (longitude) at the end of the measure
latitudeAtMeasureEnd	Double	Location (latitude) at the end of the measure
eventCode	Byte	Code of the event (see table below)

eventMeasureReceiptNumber	Long	Receipt of the measure concerned by the event
comment	String (65535)	Comments about the measure/event
tobeInvoicedSeparately	Boolean	Indicate if the measure has to be invoiced separately
sequenceCode	Byte	Sequence code running on the LMU
isBalance1	Boolean	Indicate if the Measure has been generated by the balance 1
isBalance2	Boolean	Indicate if the Measure has been generated by the balance 2
isBigContainer	Boolean	Generated measure with big container sensor
isBlack	Boolean	Notified as black when taken
isWhite	Boolean	Notified as known chip when taken
isContainer	Boolean	Generated from a container (for sequences 115/116/116 and 109 individual Ident)
isRealMeters	Boolean	Distance comes from the vehicle CANbus if true
quality	Byte	Measure quality (see table below)
slaveId	Byte	ID of the slave on the vehicle (UTI/LMU) that created the Measure

measureType	Byte	The origin of the measure:
		o=online
		w=wiga(targo)
		s=swt
		h=manually added
		x=added via web service/API (use this type with the API)
		v=weelvaarts
		m=added via kollygramcemobile

timestamp	Date	Date of the last modification on the Measure
customerId	Long	ID of the customer that owns this Measure in the database
weighingType (readonly)	String	The type of measure: 0=NotRealMeasure(Event/Emptying/Unknown) 1=Global (Is a town round, not a container) 2=Lift (Is a container taken by a lift) 3=Crane (Is a container taken by a crane)

The property “quality” gives information on the condition the Measure has been generated. Here is the description for each value:

Value	Name	Description
0	QUALITY_0	Quality 0 (best quality)
1	QUALITY_1	Quality 1
2	QUALITY_2	Quality 2
3	QUALITY_3	Quality 3
4	QUALITY_4	Quality 4
5	QUALITY_5	Quality 5
6	QUALITY_6	Quality 6 (worst quality)
7	QUALITY_Overflow	AD overflow
8	QUALITY_External	Measure not generated by an LMU
9	QUALITY_TimeOut	Time spent in the weighing gap too long
10	QUALITY_UnStable	Unstable measure
11	QUALITY_MaxSlope	Too much pitching/rolling
12	QUALITY_Overload	Maximum weight overload
13	QUALITY_Disabled	Scale disabled by a user
14	QUALITY_Disconnected	No communication with the LMU
15	QUALITY_Error	Error on the LMU

The following table lists all possible event code.

A Measure that has the property “eventCode” with a value different than 0 is in fact an **Event**.

The field “eventCode” contains the value of the first column of this table.

A brief description in three languages (French, German and English) explains what the code is used for.

The column “Can be customized” indicates if the customer is allowed to change the meaning of the event code.

The last column contains the code used in the previous version of the KOLLYgram system.

Event Code	French description	German description	English description	Can be customized	Code KG07
0	Aucun	Keiner	None	FALSE	
1	Evénement bouton LMU 115 personnalisable Nr 1	Events Taste LMU 115 anpassbar Nr 1	Customizable button event LMU 115 Nr 1	TRUE	
2	Evénement bouton LMU 115 personnalisable Nr 2	Events Taste LMU 115 anpassbar Nr 2	Customizable button event LMU 115 Nr 2	TRUE	
3	Evénement bouton LMU 115 personnalisable Nr 3	Events Taste LMU 115 anpassbar Nr 3	Customizable button event LMU 115 Nr 3	TRUE	
4	Evénement bouton LMU 115 personnalisable Nr 4	Events Taste LMU 115 anpassbar Nr 4	Customizable button event LMU 115 Nr 4	TRUE	
5	Evénement bouton LMU 115 personnalisable Nr 5	Events Taste LMU 115 anpassbar Nr 5	Customizable button event LMU 115 Nr 5	TRUE	
6	Evénement bouton LMU 115 personnalisable Nr 6	Events Taste LMU 115 anpassbar Nr 6	Customizable button event LMU 115 Nr 6	TRUE	
7	Evénement bouton LMU 115 personnalisable Nr 7	Events Taste LMU 115 anpassbar Nr 7	Customizable button event LMU 115 Nr 7	TRUE	
8	Evénement bouton LMU 115 personnalisable Nr 8	Events Taste LMU 115 anpassbar Nr 8	Customizable button event LMU 115 Nr 8	TRUE	
9	Lavage	Waschen	Wash	FALSE	
10	Erreur système RFID	RFID System Fehler	RFID System Error	FALSE	EV=1
11	Erreur de lecture RFID	Lesefehler RFID	RFID Read Error	FALSE	EV=2
12	Pas de puce RFID	Kein RFID Chip	No RFID Chip	FALSE	EV=3
13	Puce RFID défectueuse	RFID Chip kaputt	Defective RFID Chip	FALSE	EV=4
14	Puce RFID à démonter	RFID Chip abmontieren	RFID Chip to disassemble	FALSE	EV=5
15	Container vidé en plusieurs tentatives	Container mehrmals versucht zu leeren	Emptied container through many attempts	FALSE	EV=6
16	Container détérioré par le client	Container beschädigt durch Kunde	Container damaged by the customer	FALSE	EV=7
17	Container défectueux	Container kaputt	Defective container	FALSE	EV=8
18	Mauvais type de déchets	Falsche Abfallsorte	Wrong trash type	FALSE	EV=9
19	Route bloquée par de la végétation	Strasse gesperrt wegen Vegetation	Blocked road by vegetation	FALSE	EV=10

20	Route bloquée par des travaux	Strasse gesperrt wegen Bauarbeiten	Blocked road by upkeep works	FALSE	EV=11
21	Route bloquée par des véhicules	Strasse gesperrt wegen Fahrzeuge	Blocked road by vehicles	FALSE	EV=12
22	Informations diverses	Verschiedene Information	Miscellaneous informations	FALSE	DV
23	Informations du chauffeur	Information des Fahrers	Driver informations	FALSE	CH
24	Alarmes diverses	Verschiedene Alarme	Miscellaneous alert	FALSE	AL
25	Frais divers	Sonstige Kosten	Miscellaneous costs	FALSE	SP
26	Frais de carburant	Kraftstoffkosten	Fuel costs	FALSE	DI
27	Frais de péage	Mautkosten	Toll costs	FALSE	PE
28	Fin de tournée	Ende der Tour	End of the round	FALSE	FT
29	Etat des kilomètres	Kilometerstand	State of the kilometers	FALSE	KM
30	Accident	Unfall	Accident	FALSE	DE
31	Pause	Pause	Break	FALSE	PA

EventCode

It represents the type of an Event.

An EventCode contains the following properties:

Property name	Data type	Description
code	Byte, *unique*	Event code identifier, this number appears in the property "eventCode" of an Event
nameFrench	String (200)	Name of the EventCode type in French
nameGerman	String (200)	Name of the EventCode type in German
nameEnglish	String (200)	Name of the EventCode type in English
nameItalian	String (200)	Name of the EventCode type in Italian
deleted	Boolean	True if the EventCode has been deleted. False otherwise
isTask	Boolean	True if the EventCode is considered as a task. False otherwise
linkedToEventCode	Byte	Code of the EventCode this EventCode is linked to. If 0, this EventCode is not linked to another EventCode.
iconId	long	Id of the EventCodeIcon
ts_uid	long	Timestamp automatically increased on modification

EventCodeIcon

It represents the icon linked to an EventCode.

An EventCodeIcon contains the following properties:

Property name	Data type	Description
id	Byte, *unique*	Event code identifier, this number appears in the property "eventCode" of an Event
name	String(255)	Name of the icon
image	Byte[]	Binary content of the icon
ts_uid	long	Timestamp automatically increased on modification

VehicleProduct

It represents the products and the capacity of each of them that a vehicle can carry.

A VehicleProduct contains the following properties:

Property name	Data type	Description
vehicleId	Long	ID of the Vehicle in the database
productId	Long	ID of the Product in the database
capacity	Double	Capacity of this product for this vehicle [kg]
timestamp	Date	Date of the last modification on the CustomVehicleType

CustomVehicleType

A CustomVehicleType contains the following properties:

Property name	Data type	Description
Id	Long, *unique*	ID of the custom vehicle type in the database
name	String (50), *unique*	Description of the custom vehicle type that will be displayed in the screens
isTrailer	Boolean	True if this custom vehicle type represents a trailer, otherwise false
level	Long	Tells the level of the CustomVehicleType. All CustomVehicleType having a level lower or similar will be accepted either
timestamp	Date	Date of the last modification on the CustomVehicleType
customerId	Long	ID of the customer that owns this CustomVehicleType in the database

Vehicle

It represents a truck that can take Measures.

A Vehicle contains the following properties:

Property name	Data type	Description
id	Long, *unique*	ID of the Vehicle in the database
plateNumber	String (45)	Number written on the plate of the Vehicle
shortName	String (4), *unique*	Internal name for the company. (For example “V01”)
kgResolution	Decimal	Measures precision in kilograms of the vehicle
language	String (2)	Language used by the driver (DE, FR, IT, EN)
lastOnlineContact	Date	Date of the last connection by the vehicle

utiVersion	String (50)	Version number of the UTI present on the vehicle
utiVersionDate	Date	Date the UTI version has been installed on the vehicle
lmuVersion	String (50)	Version number of the LMU present on the vehicle
lmuVersionDate	Date	Date the LMU version has been installed on the vehicle
lastLatitude	Double	Last recorded latitude of the vehicle
lastLongitude	Double	Last recorded longitude of the vehicle
lastGpsDate	Date	Date of the last GPS connection on the vehicle
canRince	Boolean	True if the vehicle is able to rince containers
canWash	Boolean	True if the vehicle is able to wash containers
customVehicleTypeId	Long	Id of the custom vehicle type linked to this vehicle
supportedProductNumbers	String (255)	List of product ids separated by "," that can be delivered to this delivery point
timestamp	Date	Date of the last modification on the Vehicle
customerId	Long	ID of the customer that owns this Vehicle in the database

There are only four fields that are editable: shortName, plateNumber, customVehicleType, supportedProductNumbers. All other fields are read-only and therefore cannot be edited using PUT method.

Product

It represents a product that can be shipped by a Vehicle.

A Product contains the following properties:

Property name	Data type	Description
productNo	Byte, *unique*	Number of the product
productFrench	String (255)	Name of the product in French
productGerman	String (255)	Name of the product in German
productEnglish	String (255)	Name of the product in English
productItalian	String (255)	Name of the product in Italian
timestamp	Date	Date of the last modification on the Product
customerId	Long	ID of the customer that owns this Product in the database

Chip

It represents a chip that is installed on a container.

A Chip contains the following properties:

Property name	Data type	Description
id	Long, *unique*	ID of the Chip in the database
chipNumber	String (16), *unique*	Number of the Chip
chipTypeId	Byte	ID of the ChipType in the database
customerId	Long	ID of the customer that owns this Chip in the database
timestamp	Date	Date of the last modification on the Chip

ChipType

This is an enumeration that contains all chip types.

A ChipType contains the following properties:

Property name	Data type	Description
id	Integer, *unique*	ID of the chip type in the database
name	String (100)	Name of the chip type (same name in all languages)

Black

It represents the blacklisting history. Each record represents one modification.

A Black contains the following properties:

Property name	Data type	Description
customerId	Long	ID of the customer in the database
chipId	Long	ID of the chip in the database
black	Boolean	True if is blacklisted. False otherwise
fromWhen	Date	Will be ignored when inserting a new record Actual time stamp has to be put instead

ChipLink

It represents a link between a chip, a garbage type, a container type, a user, a location address and an invoice address.

A ChipLink contains the following properties:

Property name	Data type	Description
id	Long, *unique*	ID of the chip link in the database
chipId	Long	ID of the chip concerned in the database
customerId	Long	ID of the customer who owns this link in the database
locationAddressId	Long	ID of the location address in the database
invoiceAddressId	Long	ID of the invoice address in the database
linkedFrom	Date	Date this link is applied from
linkedTo	Date	Date this link is applied to, indeterminate if empty
deleted	Boolean	True if this link is deleted. False otherwise
garbageTypeId	Integer	ID of the garbage type in the database
containerTypeId	Integer	ID of the container type on which this chip is mounted in the database
containerNumber	String (10)	Number of the container on which this chip is mounted
Timestamp	Date	Date of the last modification on the ChipLink

ContainerType

This is an enumeration that contains all container types.

A ContainerType contains the following properties:

Property name	Data type	Description
id	Integer, *unique*	ID of the container type in the database
capacityLiters	Integer	Capacity of the container in liters
nameFrench	String (100)	Name of the garbage type in French
nameGerman	String (100)	Name of the garbage type in German
nameEnglish	String (100)	Name of the garbage type in English
nameItalian	String (100)	Name of the garbage type in Italian

GarbageType

This is an enumeration that contains all garbage types.

A GarbageType contains the following properties:

Property name	Data type	Description
id	Long, *unique*	ID of the garbage type in the database
garbageCode	String (4), *unique*	Code for the garbage (shortcut code)
nameFrench	String (100)	Name of the garbage type in French
nameGerman	String (100)	Name of the garbage type in German
nameEnglish	String (100)	Name of the garbage type in English
nameItalian	String (100)	Name of the garbage type in Italian

Address

It represents an address.
An Address can be linked to other objects.
A ChipLink and a deliveryPoint can have an invoice Address and a location Address.

An Address contains the following properties:

Property name	Data type	Description
id	Long, *unique*	ID of the Address in the database
customerId	Long	ID of the customer to whom belongs this Address in the database
streetNo	String (10)	Number of the street
additionalAddressLine	String (255)	Additional address information
name	String (255)	Name of the customer
email	String (100)	Email of the customer
fax	String (100)	Fax of the customer
phone	String (100)	Phone of the customer
contactPerson	String (255)	Contact person of the customer
remark	String (65535)	Remark concerning the Address
oldCustomerNumber	String (45)	Old customer number of the Address (May be used to store the ID of another system)
street	String (45)	Street of the Address
zip	String (10)	Postal code of the Address
city	String (45)	City of the Address

language	String (2)	Language of the customer (DE, FR, IT, EN)
custom1	String (45)	May be used depending on client usage
custom2	String (45)	May be used depending on client usage
custom3	String (45)	May be used depending on client usage
countryCode	String (2)	Country code (For example: CH, DE, FR, IT …)
timestamp	Date	Date of the last modification on the Address
addressType	Integer	Type of the address. This field is a flag: 0 = Standard address 1 = Collecting point (prediction module) 2 = Invoice address 4 = Location address 8 = Address used in the dispatching module 16 = Customer 32 = Supplier 64 = Transporter (used by internal system, do not use)

User

It represents a user of the KOLLYgram system.

A User contains the following properties:

Property name	**Data type**	**Description**
Id	Long, *unique*	ID of the User in the database
parentUserId	Long	ID of the parent of this User in the database
name	String (100), *unique*	Name of the user
userType	Integer	Value taken from the UserType
customerNumber	Integer, *unique*	Number of the customer
language	String (2)	Language of the user (DE, FR, IT, EN)
hasMessagingModule	Boolean	Is access to the messaging management enabled
hasOrdersModule	Boolean	Is access to the orders management enabled
hasKollygramModule	Boolean	Is access to the KOLLYgram enabled
hasKollytrackModule	Boolean	Is access to the KOLLYtrack enabled
hasRoadbookModule	Boolean	Is access to the Roadbook enabled
hasEventModule	Boolean	Is access to the events management enabled
accesses	List<UserAccess>	List of user access for this user

timestamp	Date	Date of the last modification on the User
customerId	Long	ID of the customer that owns this User in the database

UserType

This is an enumeration that contains all user types.

A UserType contains the following properties:

Property name	**Data type**	**Description**
id	Byte, *unique*	ID of the UserType in the database
code	String (1), *unique*	Code of the UserType (shortcut)
NameFrench	String (50)	Name of the UserType in French
NameGerman	String (50)	Name of the UserType in German
NameEnglish	String (50)	Name of the UserType in English
NameItalian	String (50)	Name of the UserType in Italian

UserAccess

It represents the rights given to a user of the KOLLYgram system.

A UserAccess contains the following properties:

Property name	**Data type**	**Description**
id	Long, *unique*	ID of the UserAccess in the database
email	String (100), *unique*	Email of the user
language	String (2)	Language of the user (DE, FR, IT, EN)
isOwner	Boolean	True if the User is a parent of another user, false otherwise
accessFrom	Date	Date from which the rights apply
accessTo	Date	Date to which the rights apply
accessToUserManagement	Boolean	Is the access to the user management enabled
accessToChipManagement	Boolean	Is the access to the chip management enabled
accessToAddressManagement	Boolean	Is the access to the address management enabled
accessToProductManagement	Boolean	Is the access to the product management enabled
accessToMeasuresImportation	Boolean	Is the access to the measures importation enabled

accessToMeasureEdition	Boolean	Is the access to the measures edition enabled
accessToMeasureInsertion	Boolean	Is the access to the measure insertion enabled
accesstoModuleGeo	Boolean	Is the access to the geo module enabled
userId	Long	ID of the user that owns these accesses in the database
firstname	String (45)	First name of the user
lastname	String (45)	Last name of the user
timestamp	Date	Date of the last modification on the UserAccess
customerId	Long	ID of the customer that owns this UserAccess in the database

Wash

It represents the wash listing history. Each record represents one modification.

A Wash contains the following properties:

Property name	Data type	Description
customerId	Long	ID of the customer in the database
chipId	Long	ID of the chip in the database
toWash	Boolean	True if needs to be washed. False otherwise
fromWhen	Date	Will be ignored when inserting a new record Actual time stamp has to be put instead

Rince

It represents the rince listing history. Each record represents one modification.

A Rince contains the following properties:

Property name	Data type	Description
customerId	Long	ID of the customer in the database
chipId	Long	ID of the chip in the database
toRince	Boolean	True if needs to be rinsed. False otherwise
fromWhen	Date	Will be ignored when inserting a new record Actual time stamp has to be put instead

CustomDeliveryPointType

It represents a personalized type of geographical location that are specific Delivery, Loading, Neutral or Delivery and Loading points type.

A CustomDeliveryPointType contains the following properties:

Property name	Data type	Description
id	Long, *unique*	ID of the custom delivery point type in the database
name	String (50)	Description of the custom delivery point type that will be displayed in the screens
kind	Long	Kind of the delivery point custom type. Can be one of those: 0 = Delivery 1 = Loading 2 = Neutral 3 = Delivery + Loading
timestamp	Date	Date of the last modification on the CustomDeliveryPointType
customerId	Long	ID of the customer that owns this CustomDeliveryPointType in the database

DeliveryPoint

It represents a geographical location to which we can plan an order for.

A DeliveryPoint contains the following properties:

Property name	Data type	Description
id	Long, *unique*	ID of the delivery point in the database
customerId	Long	ID of the transporter in the database
realCustomerId	Long	ID of the customer in the database
customDeliveryPointTypeId	Long	ID of the custom delivery point type linked to this delivery point
externalCustomerId	String (20)	ID of the customer in the outside servers of the customer
info1	String (50)	Main identification of the delivery point. Used in the screen where a delivery point must be displayed
info2	String (50)	First secondary identification of the delivery point
info3	String (50)	Second secondary identification of the delivery point
info4	String (50)	Third secondary identification of the delivery point
deliveryAddressId	Long	Id of the address where the truck must deliver for this delivery point

invoiceAddressId	Long	Id of the address where the invoice must be sent to
officeRemark	String (255)	Remark for the delivery point that only people of the office can read
internalRemark	String (255)	Remark for the delivery point that only internal people can read
reportRemark	String (255)	Remark for the delivery point that will appear on the delivery report given to the customer
latitude	Double	Latitude of the delivery point
longitude	Double	Longitude of the delivery point
baseDeliveryTheoricalTime	Double	Base time needed to proceed a delivery for this point. Correspond to the time needed if there is a delivery quantity of 0 [minutes]
perUnitDeliveryTheoricalTime	Double	Time needed to deliver a unit of product (generally 1 [kg]) [minutes]
cleaningTheoricalTime	Double	Cleaning time [minutes]
supportedProductNumbers	String (255)	List of product ids separated by "," that can be delivered to this delivery point
supportedVehicleTypeIds	String (255)	List of vehicle type ids separated by "," that this delivery point can receive

periodType	String (50)	If string is not empty, then the DeliveryPoint is a periodic order. The periodicity is defined with this structure:
		
		<nbDays>D = days
		W = week
		M = month
		Q = quarter
		H = half-year
		Y = year
		
		Samples: 3M, 6D, 4W, 1Y (every 3 months, 6 days, 4 weeks, 1 year)

timestamp	Date	Date of the last modification on the DeliveryPoint
pipesLength	double	Length of the pipes [m]

Additional fields in DeliveryPoint for consumption predictive algorithm:

Property name	**Data type**	**Description**
annualConsumption	Double	Annual average consumption for this delivery point [kg/year]
consumptionCoefficient	Double	Coefficient of correction of the average consumption for this delivery point [%]
capacity	Double	Maximal capacity of the delivery point [kg]
lastDelivery	Date	Date of the last time the delivery point has been delivered
lastOrderMissed	Boolean	True if the client linked to this delivery point did not order by us the last time it was filled in
lastKnownFillingRate	Double	The percentage of filling of the silo right after the last delivery (usually 100%)
stationId	Long	The linked Meteorological station
useNearestStation	Boolean	If true, calculates the nearest meteorological station based on the latitude and longitude, otherwise use stationId
power	Double	The power of the heater [kWh]

DeliveryPointFilling

It represents a previous filling for a DeliveryPoint.

A DeliveryPointFilling contains the following properties:

Property name	**Data type**	**Description**
id	Long, *unique*	ID of the DeliveryPointFilling in the database
deliveryPointId	Long	ID of the DeliveryPoint that has been filled
percentage	Double	Percentage of filling after the delivery
fillingDate	Date	Date of the last filling
capacity	Double	Full capacity of the DeliveryPoint at the time of the filling
source	long	The source that set this filling
timestamp	Date	Date of the last modification on the DeliveryPointFilling
customerId	Long	ID of the customer that owns this DeliveryPointFilling in the database

Order

An Order represents a mission to be executed by a vehicle for a specific delivery point.

The order type is defined by the value of the quantity:

- 0 = neutral
- < 0 = Delivery
- > 0 = Loading

It contains the following properties:

Property name	Data type	Description
id	Long, *unique*	ID of the Order in the database
externalOrderId	Long	ID of the Order in the external servers of the customer
deliveryPointId	Long	Id of the delivery point to which product must be delivered
plannedVehicleId	Long	Id of the vehicle planned for the realization of this order
deliveringVehicleId (readonly)	Long	Id of the vehicle that effectively realized this order
productId	Long	The id of the product that must be delivered
plannedOn	Date	Planned date for the realization of this order
plannedPosition	Long	Position of the order in the order's chronological list to be done by the vehicleId on the plannedOn day (1-base chronological order)
deliveredOn (readonly)	Date	Effective realization of this order
plannedQuantity	Double	Quantity to be delivered [Kg]
effectiveQuantity (readonly)	Double	Quantity effectively delivered [Kg]
internalRemark	String (255)	Remark for the order that only internal people can read (taken by default from delivery point)
reportRemark	String (255)	Remark for the order that will appear on the delivery report given to the customer (taken by default from delivery point)
driverRemark (readonly)	String (255)	Remark written by the driver after the delivery
receiptNumber (readonly)	Long	The number of the receipt given to the customer
batchNumber	String (10)	The batch number of the delivered product
stillToBeDeliveredQuantity (readonly)	Double	The quantity that must still be delivered in a new order
startDelay	Date	Starting deadline (date and time) if delayPrecision is set as P
endDelay	Date	Ending deadline (date and time) if delayPrecision is set as P

delay	Date	Deadline (date and time). Must be combine with delayPrecision.

delayPrecision	String (1)	The precision applied to the planned date for the delivery time. Value can be:
		
		H = hour
		I = half-day
		D = day
		W = week
		M = month
		X = no delay
		P = period

fillingRate (readonly)	Double	The silo filling rate after the delivery has been done
status	Long	The status of the order. One of those values: 0 = new 1 = completed 2 = not completed 3 = cancelled
timestamp	Date	Date of the last modification on the Order
unit	String(50)	The unit of the delivered quantity. Can be various units (M3, Kg, ….)
customerId	Long	ID of the customer that owns this Order in the database
blowDurationMinutes (readonly)	Integer	Number of minutes that was needed to fill the DeliveryPoint
aspirationDurationMinutes (readonly)	Integer	Number of minutes that was needed to empty the DeliveryPoint
aspirationPressure (readonly)	Double	Pressure of the aspiration [bar]
linkedMeasureId (readonly)	Long (nullable)	Id of the measure that executed the order. Is null if the order is not executed or if the order has no corresponding measure.

MeteorologicalStation (readonly)

It represents a meteorological station used to calculate temperature differences coefficient in consumption predictive algorithm.

A MeteorologicalStation contains the following properties:

Property name	**Data type**	**Description**
id	Long, *unique*	ID of the meteorological station
name	String (50)	Name of the meteorological station
altitude	Double	Altitude of the meteorological station

latitude	Double	Latitude of the meteorological station
longitude	Double	Longitude of the meteorological station
code	String (3)	Code of the meteorological station

IdentChip (customer idents)

It represents a chip which is sent to the vehicles in order to identify the detected chips.

If the option "Use Customer Idents" is set for the transporter (available on KOCO-online), the idents generated by the system will be ignored and those IdentChips will be used instead.

An IdentChip contains the following properties:

Property name	Data type	Description
Key	String (16)	Number of the chip. Must be a hexadecimal number.
ContainerNumber	String (50)	Number written on the container
ContainerType	String (50)	Type and capacity of the container
CustomerNumber	String (50)	Number of the customer to which the chip has been assigned
GarbageType	String (50)	Type of garbage inside the container
CustomerLastName	String (50)	Last name of the customer
CustomerFirstName	String (50)	First name of the customer
AdditionalInfos	String (50)	Additional info concerning the customer and his address
Street	String (50)	Customer street
ZIP	String (50)	Customer ZIP (postal number)
Locality	String (50)	Customer town
IsBlack	Boolean	True if the customer is blacklisted. False otherwise.

IdentCustomer (customer idents)

It represents a customer and its information which are sent to the vehicles in order to link measures to a customer.

If the option "Use Customer Idents" is set for the transporter (available on KOCO-online), the idents generated by the system will be ignored and those IdentCustomers will be used instead.

An IdentCustomer contains the following properties:

Property name	Data type	Description
Key	String (10)	Customer number
CustomerName	String (50)	Name of the customer/town

Street	String (50)	Customer street
ZIP	String (50)	Customer ZIP (postal number)
Locality	String (50)	Customer town

Controllers

This chapter will iterate the controllers and their actions that a client can use in order to achieve modifications on the server.

A controller is an open gate that clients can access through the web. A controller offers functions that will modify data in the database.

There are four functionalities a controller can propose:

GET:	A function of type GET will search data in the database that satisfy the request and send them back to the client.
POST:	A function of type POST is used to store a model in the database.
PUT:	A function of type PUT is used to modify an existing model already stored in the database.
DELETE:	A function of type DELETE is used to delete an existing model in the database.

When a function of a controller is called, many controls are executed before the database is modified to verify that the request is correct and to check that the current user possesses the mandatory rights to execute the request.

If one of the mentioned controls failed, a message that indicates the reason of the failure is sent back to the client.

If the request is executed successfully, the controller transfers the result to the client.

Note: Not all controllers have every four functionalities depending on the business logic. Similarly, some controllers have more than one function by functionality.

Take and off options

Some actions can accept filters such as “take” and “off”.

The “take” option will limit the result to the amount given as parameter.

The “off” option will skip the first *n* records.

Example: /Measures?take=500&off=100

This example will skip the first 100 records and retrieve the next 500 ones.

It will be written in the documentation below if these options are allowed.

The timestamp property

The following models have a “timestamp” property:

- Address
- Measure
- User
- UserAccess
- Product
- Chip
- ChipLink
- VehicleProduct
- CustomVehicleType
- Vehicle
- CustomDeliveryPointType
- DeliveryPoint
- DeliveryPointFilling
- Order

This property indicates the last precise time the model has been modified (accurate to the second).

In addition, their controller offers an action that retrieve all models that possesses a timestamp similar or more recent than the parameter.

For example, the URL “/Measures/ts/**2015**07**13**15**04**00” will list all Measures with a timestamp equal or more recent than the 13th of July 2015 at 15:04:00.

Thanks to this, it’s easy to synchronize a client application with the latest modifications made on the KOLLYgram system.

Notice: the other models don’t have this property because they are either enumerations, either history records.

The following section will explain in further details the functions of each controller.

MeasuresController

This is the controller that handles requests concerning Measures.

Only Measures can be managed through this controller. For Events, see EventsController.

The MeasuresController offers the following functions:

Functionality	URL	Description
GET	/measures [?take=2000&off=1000]	Retrieve all Measures of the current day. *You can use take and off options.*
GET	/measures/date/20150308 [?take=2000&off=1000]	Retrieve all Measures of the specified day (the date format must be yyyyMMdd) *You can use take and off options.*
GET	/measures/fromId/1000 [?take=2000&off=1000 &toBeInvoicedSeparately=null]	Retrieve all Measures that possess a greater ID than the given one. Optional parameters: **toBeInvoicedSeparately** : - null => return all measures (default) - true => only return measures to invoice - false => only return measures not to invoice
GET	/measures/ts/20150713144852 [?take=2000&off=1000]	Retrieve all Measures with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss) The retrieved Measures can be new or modified. *You can use take and off options.*
GET	/measures/perVehicle /{vehicleId}/20160120 [?seeContainer=true &seeStandard=true &seePublicWeight=true &seeEmptying=true &seeDeleted=false &take=2000 &off=1000]	Retrieve all Measures generated by a Vehicle on the given date (the date format must be yyyyMMdd) Optional parameters: **seeContainer** = allow Measures with containers (default: true) **seeStandard** = allow global Measures (default: true) **seePublicWeight** = allow Measures from dumps (default: true) **seeEmptying** = allow Measures from truck emptying (default: true) **seeDeleted** = allow deleted Measures (default: false) *You can use take and off options.*
GET	/measures/perPeriod /201601010000/201601201149 [?owner=123 &vehicleIds=10,11,12 &seeContainer=true &seeStandard=true &seePublicWeight=true &seeEmptying=true &seeDeleted=false &onlyBiggerThanZero=false &chipFilter=2 &take=2000 &off=1000]	Retrieve all Measures between the first and the second date (the dates format must be yyyyMMddHHmm) Optional parameters: **owner** = only Measures with this customer number **vehicleIds** = only Measures from these Vehicle Ids (comma separated) **seeContainer** = allow Measures with containers (default: true) **seeStandard** = allow global Measures (default: true) **seePublicWeight** = allow Measures from dumps (default: true) **seeEmptying** = allow Measures from truck emptying (default: true) **seeDeleted** = allow deleted Measures (default: false) **onlyBiggerThanZero** = only Measures with netWeight > 0 (default: false) **chipFilter** = 1-Only with chip, 2-Only without chip, 3-Only known chip, 4-Only unknown chip, other=All *You can use take and off options.*
GET	/measures/10751	Retrieve the Measure with the given ID
POST	/measures	Insert a new Measure
PUT	/measures/10751	Modify the Measure with the given ID
DELETE	/measures/10751	Delete the Measure with the given ID

EventsController

This is the controller that handles requests concerning Events.

Only Events can be managed through this controller. For Measures, see MeasuresController

The EventsController offers the following functions:

Functionality	URL	Description
GET	/events [?take=2000&off=1000]	Retrieve all Events of the current day. *You can use take and off options.*
GET	/events/date/20150308 [?take=2000&off=1000]	Retrieve all Events of the specified day (the date format must be yyyyMMdd) *You can use take and off options.*
GET	/events/fromId/1000 [?take=2000&off=1000]	Retrieve all Events that possesses a greater ID than the given one
GET	/events/ts/20150713144852 [?take=2000&off=1000]	Retrieve all Events with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss) *You can use take and off options.*
GET	/events/perPeriod/20160101/20160120 [?owner=123 &vehicleIds=1,2,3 &eventFilter=2 &eventCodes=9,10,11,23 &seeDeleted=false &take=2000 &off=1000]	Retrieve all Events between the first and the second date (the dates format must be yyyyMMdd) Optional parameters: **owner** = only Events with this customer number **vehicleIds** = only Events from these Vehicle Ids (comma separated) **eventFilter** = 1-Only with task, 2-Only with opened task, 3-Only with closed task, other=All **eventCodes** = only Events with these event codes (comma separated) **seeDeleted** = allow deleted Events (default: false) *You can use take and off options.*
GET	/events/10751	Retrieve the Event with the given ID
POST	/events	Insert a new Event
PUT	/events/10751	Modify the Event with the given ID
DELETE	/events/10751	Delete the Event with the given ID

EventCodesController

This is the controller that handles requests concerning EventCodes.

The EventCodesController offers the following functions:

Functionality	URL	Description
GET	/eventcodes	Retrieve all EventCodes.

EventCodeIconsController

This is the controller that handles requests concerning EventCodeIcons.

The EventCodeIconsController offers the following functions:

Functionality	URL	Description
GET	/eventcodeicons	Retrieve all EventCodeIcons.

CustomVehicleTypesController

This is the controller that handles requests concerning CustomVehicleTypes.

The CustomVehicleTypesController offers the following functions:

Functionality	URL	Description
GET	/customVehicleTypes [?take=2000&off=1000]	Retrieve all CustomVehicleTypes belonging to the current user
GET	/customVehicleTypes/ts/ 20160622145232 [?take=2000&off=1000]	Retrieve all CustomVehicleTypes with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss)
GET	/customVehicleTypes/fromLevel/ 4 [?take=2000&off=1000]	Retrieve all CustomVehicleTypes with a level below or equal to the one specified
GET	/customVehicleTypes/5425	Retrieve the CustomVehicleType with given ID
POST	/customVehicleTypes	Insert a new CustomVehicleType
PUT	/customVehicleTypes/4324	Modify the CustomVehicleType with the given ID Note: only the carrier can modify a CustomVehicleType
DELETE	/customVehicleTypes/3214	Delete the CustomVehicleType with the given ID if not linked to a vehicle

VehiclesController

This is the controller that handles requests concerning Vehicles.

The VehiclesController offers the following functions:

Functionality	URL	Description
GET	/vehicles [?take=2000&off=1000]	Retrieve all Vehicles belonging to the carrier of the current User
GET	/vehicles/ts/20150713145232 [?take=2000&off=1000]	Retrieve all Vehicles with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss)
GET	/vehicles/5310	Retrieve the Vehicle with given ID
PUT	/vehicles/5310	Modify the Vehicle with the given ID Note: only the carrier can modify a Vehicle There are only two fields that are editable: shortName, plateNumber. All other fields are read-only and therefore cannot be edited using PUT method.

ProductsController

This is the controller that handles requests concerning Products.

The ProductsController offers the following functions:

Functionality	URL	Description
GET	/products [?take=2000&off=1000]	Retrieve all Products belonging to the current User and his children
GET	/products/ts/20150713145232 [?take=2000&off=1000]	Retrieve all Products with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss)
GET	/products/8	Retrieve the Product with the given ID
PUT	/products/8	Insert or modify the Product with the given ID

ChipsController

This is the controller that handles requests concerning Chips.

The ChipsController offers the following functions:

Functionality	URL	Description
GET	/chips [?take=2000&off=1000]	Retrieve all Chips belonging to the current User and his children. *You can use take and off options.*
GET	/chips/ts/20150713145332 [?take=2000&off=1000]	Retrieve all Chips with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss)
GET	/chips/110810	Retrieve the Chip with the given ID
GET	/chips/nr/000FFFFFFAE	Retrieve the chip with the given number
POST	/chips	Insert a new Chip
PUT	/chips/110810	Modify the Chip with the given ID
DELETE	/chips/110810	Delete the Chip with the given ID

ChipTypesController

This is the controller that handles requests concerning ChipTypes.

The ChipTypesController offers the following functions:

Functionality	URL	Description
GET	/chiptypes	Retrieve all ChipTypes

BlacksController

This is the controller that handles requests concerning Blacks.

The BlacksController offers the following functions:

Functionality	URL	Description
GET	/blacks [?take=2000&off=1000]	Retrieve the current blacklist (list of all Chips currently blacklisted). *You can use take and off options.*
GET	/blacks/35105	Retrieve the blacklist history for the given Chip ID
POST	/blacks	Mark a Chip as black or white from now on

ChipLinksController

This is the controller that handles requests concerning ChipLinks.

The ChipLinksController offers the following functions:

Functionality	URL	Description
GET	/chiplinks [?take=2000&off=1000]	Retrieve all ChipLinks belonging to the current User and his children
GET	/chiplinks/chip/24041	Retrieve all ChipLinks concerned by the given Chip ID
GET	/chiplinks/ts/20150713145432 [?take=2000&off=1000]	Retrieve all ChipLinks with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss)
GET	/chiplinks/15101	Retrieve the ChipLink with the given ID
POST	/chiplinks	Create a new ChipLink
PUT	/chiplinks/15101	Modify the ChipLink with the given ID
DELETE	/chiplinks/15101	Mark the ChipLink with the given ID as deleted

ContainerTypesController

This is the controller that handles requests concerning ContainerTypes.

The ContainerTypesController offers the following functions:

Functionality	URL	Description
GET	/containertypes	Retrieve all ContainerTypes

GarbageTypesController

This is the controller that handles requests concerning GarbageTypes.

The GarbageTypesController offers the following functions:

Functionality	URL	Description
GET	/garbagetypes	Retrieve all GarbageTypes

AddressesController

This is the controller that handles requests concerning Addresses.

The AddressesController offers the following functions:

Functionality	URL	Description
GET	/addresses [?take=2000&off=1000]	Retrieve all Addresses belonging to the current User and his children. *You can use take and off options.*
GET	/addresses/ts/20150713145432 [?take=2000&off=1000]	Retrieve all Addresses with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss)
GET	/addresses/16541	Retrieve the Address with the given ID
POST	/addresses	Create a new Address
PUT	/addresses/16541	Modify the Address with the given ID
DELETE	/addresses/16541	Delete the Address with the given ID

UsersController

This is the controller that handles requests concerning Users.

The UsersController offers the following functions:

Functionality	URL	Description
GET	/users [?take=2000&off=1000]	Retrieve all Users belonging to the current User and his children
GET	/users/ts/20150713145432 [?take=2000&off=1000]	Retrieve all Users with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss)
GET	/users/29841	Retrieve the User with the given ID
POST	/users	Create a new User
PUT	/users/29841	Modify the User with the given ID
DELETE	/users/29841	Delete the User with the given ID

UserTypesController

This is the controller that handles requests concerning UserTypes.

The UserTypesController offers the following functions:

Functionality	URL	Description
GET	/usertypes	Retrieve all UserTypes

UserAccessesController

This is the controller that handles requests concerning UserAccesses.

The UserAccessesController offers the following functions:

Functionality	URL	Description
GET	/useraccesses [?take=2000&off=1000]	Retrieve all UserAccesses belonging to the current User *You can use take and off options.*
GET	/useraccesses/ts/20150713145532 [?take=2000&off=1000]	Retrieve all UserAccesses with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss)
GET	/useraccesses/user/18051	Retrieve all UserAccesses belonging to the given User and his children.
GET	/useraccesses/59516	Retrieve the UserAccess with the given ID

ToolsController

This is a controller that provides various services. No models are specifically used with this controller.

This controller offers the following functions:

Functionality	URL	Description
GET	/GetUTCServerTime	Retrieve the current time of the server Can be useful if a synchronization is needed

WashesController

This is the controller that handles requests concerning Washes.

The WashesController offers the following functions:

Functionality	URL	Description
GET	/washes [?take=2000&off=1000]	Retrieve the current wash list (list of all Chips currently wash listed). *You can use take and off options.*
GET	/washes/35105	Retrieve the wash list history for the given Chip ID
POST	/washes	Mark a Chip as “to wash” or not from now on

RincesController

This is the controller that handles requests concerning Rinces.

The RincesController offers the following functions:

Functionality	URL	Description
GET	/rinces [?take=2000&off=1000]	Retrieve the current rinse list (list of all Chips currently rinse listed). *You can use take and off options.*
GET	/rinces/35105	Retrieve the rinse list history for the given Chip ID
POST	/rinces	Mark a Chip as “to rinse” or not from now on

CustomDeliveryPointTypesController

This is the controller that handles requests concerning CustomDeliveryPointTypes.

The CustomDeliveryPointTypesController offers the following functions:

Functionality	URL	Description
GET	/customDeliveryPointTypes [?take=2000&off=1000]	Retrieve all CustomDeliveryPointTypes belonging to the current User
GET	/customDeliveryPointTypes/ts/ 20160622145232 [?take=2000&off=1000]	Retrieve all CustomDeliveryPointTypes with a timestamp equal or more recent than the given date. (The date format must be yyyyMMddHHmmss)
GET	/customDeliveryPointTypes/5425	Retrieve the CustomDeliveryPointType with given ID
POST	/customDeliveryPointTypes	Insert a new CustomDeliveryPointType
PUT	/customDeliveryPointTypes/4324	Modify the CustomDeliveryPointType with the given ID Note: only the carrier can modify a delivery point type
DELETE	/customDeliveryPointTypes/3214	Delete the CustomDeliveryPointType with the given ID if not linked to a delivery point

DeliveryPointsController

This is the controller that handles requests concerning DeliveryPoints.

The DeliveryPointsController offers the following functions:

Functionality	URL	Description

GET	/deliveryPoints
	[?stateIds=2,3,4
	vehicleTypeIds=1,3,4
	productNumbers=1,5,6
	genericTypeIds=2,3,4
	customizedTypeIds=1,3,4
	&take=2000
	&off=1000]
		Retrieve all DeliveryPoints belonging to the current User and his children.
		stateIds = only DeliveryPoints having these states (comma separated)
		0 = No Info
		1 = Planned
		2 = To be delivered
		3 = To be delivered soon
		4 = To be contacted
		5 = Nothing to do
		vehicleTypeIds = only DeliveryPoints supporting these Vehicle Type Ids (comma separated)
		productNumbers = only DeliveryPoints supporting those product numbers (comma separated)
		genericTypeIds = only DeliveryPoints of these category (comma separated)
		0 = Delivery
		1 = Loading
		2 = Neutral
		3 = Delivery + Loading

		customizedTypeIds = only DeliveryPoints of these customized deliveryPoint types (comma separated) *You can use take and off options.*

GET	/deliveryPoints/ts/20160622145232
	[?stateIds=2,3,4
	vehicleTypeIds=1,3,4
	productNumbers=1,5,6
	genericTypeIds=2,3,4
	customizedTypeIds=1,3,4
	&take=2000
	&off=1000]
		Retrieve all DeliveryPoints with a timestamp equal or more recent than the given date.
		(The date format must be yyyyMMddHHmmss)
		stateIds = only DeliveryPoints having these states (comma separated)
		0 = No Info
		1 = Planned
		2 = To be delivered
		3 = To be delivered soon
		4 = To be contacted
		5 = Nothing to do
		vehicleTypeIds = only DeliveryPoints supporting these Vehicle Type Ids (comma separated)
		productNumbers = only DeliveryPoints supporting those product numbers (comma separated)
		genericTypeIds = only DeliveryPoints of these category (comma separated)
		0 = Delivery
		1 = Loading
		2 = Neutral
		3 = Delivery + Loading
		customizedTypeIds = only DeliveryPoints of these customized deliveryPoint types (comma separated)
		
		You can use take and off options.

GET	/deliveryPoints/5425	Retrieve the DeliveryPoint with given ID
POST	/deliveryPoints	Insert a new DeliveryPoint
PUT	/deliveryPoints/4324	Modify the DeliveryPoint with the given ID Note: only the carrier can modify a delivery point
DELETE	/deliveryPoints/3214	Delete the DeliveryPoint with the given ID

OrdersController

This is the controller that handles requests concerning Orders.

The OrdersController offers the following functions:

Functionality	URL	Description
GET	/orders [?take=2000 &off=1000]	Retrieve the orders list (list of all Orders). *You can use take and off options.*
GET	/orders/TodoInDays/90 [?onlyUnassigned=true &vehicleTypeIds=1,3,5 &orderTypeIds=4,5,7 &productNumbers=1,3,9 &orderStatusIds=3,5,6 &take=2000 &off=1000]	Retrieve the orders to be done in the next x days (here 90 days) Optional parameters: **unassigned** = only Orders not assigned yet to a vehicle **vehicleTypeIds** = only Orders from these Vehicle Type Ids (comma separated) **orderTypeIds** = only Orders with quantity corresponding to the specified order types list (comma separated) 1 = Delivery (Quantity <0) 2 = Neutral (Quantity =0) 3 = Loading (Quantity >0) **productNumbers** = only Orders linked to those product numbers (comma separated) **orderStatus** = only Orders with this status ids (comma separated), see Order model

		You can use take and off options.
GET	/orders/Done/20160401/20160630 &vehicleTypeIds=1,3,5 &orderTypeIds=4,5,7 &productNumbers=1,3,9 &orderStatusIds=3,5,6 &take=2000 &off=1000]	Retrieve the orders done in the specified time frame. Here, 20160401 is the start date, 20160630 is the end date (the date format must be yyyyMMdd) Optional parameters: **unassigned** = only Orders not assigned yet to a vehicle **vehicleTypeIds** = only Orders from these Vehicle Type Ids (comma separated) **orderTypeIds** = only Orders with quantity corresponding to the specified order types list (comma separated) 1 = Delivery (Quantity <0) 2 = Neutral (Quantity =0) 3 = Loading (Quantity >0) **productNumbers** = only Orders linked to those product numbers (comma separated) **orderStatus** = only Orders with this status ids (comma separated), see Order model *You can use take and off options.*
GET	/orders/Periodic [?onlyUnassigned=true &vehicleTypeIds=1,3,5 &orderTypeIds=4,5,7 &productNumbers=1,3,9 &orderStatusIds=3,5,6 &take=2000 &off=1000]	Retrieve the periodical orders Optional parameters: **unassigned** = only Orders not assigned yet to a vehicle **vehicleTypeIds** = only Orders from these Vehicle Type Ids (comma separated) **orderTypeIds** = only Orders with quantity corresponding to the specified order types list (comma separated) 1 = Delivery (Quantity <0) 2 = Neutral (Quantity =0) 3 = Loading (Quantity >0) **productNumbers** = only Orders linked to those product numbers (comma separated) **orderStatus** = only Orders with this status ids (comma separated), see Order model *You can use take and off options.*
GET	/orders/Planned/321/20160504 [?orderTypeIds=4,5,7 &productNumbers=1,3,9 &orderStatusIds=3,5,6 &take=2000 &off=1000]	Retrieve the orders planned for the specified vehicleId on the specified planificationDate (the date format must be yyyyMMdd) Here 321 is the vehicle id, 20160504 is the planification date Optional parameters: **unassigned** = only Orders not assigned yet to a vehicle **vehicleTypeIds** = only Orders from these Vehicle Type Ids (comma separated) **orderTypeIds** = only Orders with quantity corresponding to the specified order types list (comma separated) 1 = Delivery (Quantity <0) 2 = Neutral (Quantity =0) 3 = Loading (Quantity >0) **productNumbers** = only Orders linked to those product numbers (comma separated) **orderStatus** = only Orders with this status ids (comma separated), see Order model *You can use take and off options.*
GET	/orders/ts/20160622145232 [?onlyUnassigned=true	Retrieve all Orders with a timestamp equal or more recent than the given date.

	&vehicleTypeIds=1,3,5 &orderTypeIds=4,5,7 &productNumbers=1,3,9 &orderStatusIds=3,5,6 &take=2000 &off=1000]	(The date format must be yyyyMMddHHmmss) Optional parameters: **unassigned** = only Orders not assigned yet to a vehicle **vehicleTypeIds** = only Orders from these Vehicle Type Ids (comma separated) **orderTypeIds** = only Orders with quantity corresponding to the specified order types list (comma separated) 1 = Delivery (Quantity <0) 2 = Neutral (Quantity =0) 3 = Loading (Quantity >0) **productNumbers** = only Orders linked to those product numbers (comma separated) **orderStatus** = only Orders with this status ids (comma separated), see Order model *You can use take and off options.*
GET	/orders/35105	Retrieve the order with the corresponding ID
POST	/orders	Insert a new Order
PUT	/orders/4324	Modify the Order with the given ID Note: only the carrier can modify an order
DELETE	/orders/3214	Delete the Order with the given ID

MeteorologicalStationsController

This is the controller that handles requests concerning MeteorologicalStations.

The MeteorologicalStationsController offers the following functions:

Functionality	URL	Description
GET	/meteorologicalStations	Retrieve all meteorological stations

IdentChipsController

This is the controller that handles requests concerning IdentChips.

The IdentChipsController offers the following functions:

Functionality	URL	Description
POST	/IdentChips	Update the ident chips list (customer idents). New ident key → ident added Existing ident key → ident updated Missing ident key → ident deleted

IdentCustomersController

This is the controller that handles requests concerning IdentCustomers.

The IdentCustomersController offers the following functions:

Functionality	URL	Description
POST	/IdentCustomers	Update the ident customers list (customer idents). New ident key → ident added Existing ident key → ident updated Missing ident key → ident deleted

Web access

This chapter will explain how to consume the API through web access.

Like explained sooner, all controllers offer their functions on the web.

In order to consume the services they propose, the client (human and/or application) must use a URI (Unified Resource Identifier) that is specific to each function of each controller.

For example, in order to retrieve all ChipLinks concerned by a Chip ID, the corresponding URI is:

www.kollygramserver.ch/chiplinks/chip/10751

The first part of the URI is the name of the server. The next part is the name of the controller. It's always the name of the controller without "Controller": (UsersAccessesController → UserAccesses). The last part is specific to the function. Every functions and their URI are listed in the previous chapter.

Note: the use of capital letters has no influence on the routing of the request.

Another element to keep in mind is the HTTP method corresponding to the URI. For instance, the request must include the keyword "GET" before the URI into the request. If this step is omitted, the function won't be found.

Finally, when inserting or updating a model, the body of the request must contain a JSON (JavaScript Object Notation) serialization of the object.[1] If the request has no body or the JSON standard is not respected, the controller will advise the client that something is missing.

If the request is correct, the controller will return an answer to the client. If the expected answer is a model or a list of models, the body of the response will contain a JSON instance that can be deserialized by the client. In other cases, the response is plain text. For example, when deleting a model, the controller will return "true" if the action succeeded. Likewise when getting the current time of the server.

[1] See Appendix 1

Response status code

The following table explains the responses given by the server.

Responses OK

200	OK	General status code. Most common code used to indicate success.
201	CREATED	Successful creation occurred (via either POST or PUT). Set the Location header to contain a link to the newly-created resource (on POST). Response body content may or may not be present.
204	NO CONTENT	Indicates success but nothing is in the response body, often used for DELETE and PUT operations

Responses with errors on the client request

400	BAD REQUEST	General error when fulfilling the request would cause an invalid state. Domain validation errors, missing data, etc. are some examples.
401	UNAUTHORIZED	Error code response for missing or invalid authentication token.
403	FORBIDDEN	Error code for user not authorized to perform the operation or the resource is unavailable for some reason (e.g. time constraints, etc.).
404	NOT FOUND	Used when the requested resource is not found, whether it doesn't exist or if there was a 401 or 403 that, for security reasons, the service wants to mask
405	METHOD NOT ALLOWED	Used to indicate that the requested URL exists, but the requested HTTP method is not applicable. For example, POST /users/12345 where the API doesn't support creation of resources this way (with a provided ID). The Allow HTTP header must be set when returning a 405 to indicate the HTTP methods that are supported. In the previous case, the header would look like "Allow: GET, PUT, DELETE"
409	CONFLICT	Whenever a resource conflict would be caused by fulfilling the request. Duplicate entries, such as trying to create two customers with the same information, and deleting root objects when cascade-delete is not supported are a couple of examples.

Responses with errors on the server side

500	INTERNAL SERVER ERROR	Never return this intentionally. The general catch-all error when the server-side throws an exception. Used only for errors that the consumer cannot address from their end

Authentication

This chapter will explain how the authentication system works.

The KOLLYgram system is a secured system. The database contains private data that aren't accessible for non-authorized users.

Therefore, in order to use the KOLLYgram system, an account is mandatory. Every customer of the KOLLYgram system have an account that they can use to log on the web site for example.

This account gives access to some modules. A module is a set of functionalities on the KOLLYgram system.

Every users who are part of this account receive a user access that inherits of the parent account rights. This user access specifies in details which rights the user possesses, the language of the user and other options.

In addition to the standard KOLLYgram account, an API access is needed to use the services of the API. This access can be created for the customers of the KOLLYgram system who need to access their data in a different way.

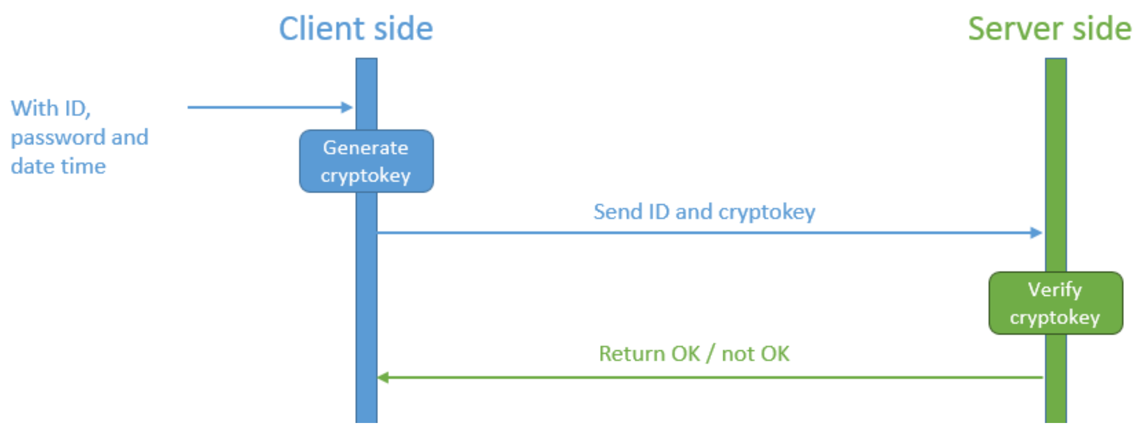
Note: some of the API controllers may be denied if the KOLLYgram account hasn't the required modules enabled.

The API access is linked to a user and get the same rights.
This access has a validity period. The access is denied if it doesn't occur within this validity range.
Finally, every API access gets a generated unique key that is used as a password.

In order to go through the authentication system, a request must include two head fields that will help the server to authenticate the user that sent the request:

x-kollygram-client: kolly-123456
x-kollygram-signature: eDppF0JtFGPKlJC9XksR23B3rSA=

The first field is the ID of the user access that possess this API access in the database.
The second field is the unique key/password that has been encrypted with the current time of the request.



Here is the algorithm to generate a cryptokey from the ID and password:

The text in green are new variables.
The text in orange are variables entered by the user (login).

1. Stock the current date and time with the format yyyyMMddHHmm in a variable → ts
2. Stock the text “kgua-“ preceding the ID in a variable → client
3. Create an ASCIIEncoding object
4. Replace “-“ with “+” and “_” with “/” in the password
5. Create an array of bytes with the password in Base 64→ privateKeyBytes
6. Create an array of bytes by using the ASCIIEndoding object on ts and client together → encodedTsAndClientBytes
7. Create a HMACSHA1 object thanks to privateKeyBytes → algorithm
8. Create an array of bytes by passing encodedTsAndClientBytes through algorithm → hash
9. Create a string from base 64 with hash and replace “+” with “-“ and “/” with “_”→ signature
10. The cryptokey is signature

In addition to that algorithm, here is a sample code in VB.NET and in C#:

VB.NET

```
Dim ts = Date.Now.ToString(« yyyyMMddHHmm »)
Dim client = “kgua-“ & UserAccessId
Dim encoding As ASCIIEncoding = New ASCIIEncoding()
Dim privateKeyBytes As Byte() = Convert.FromBase64String(password.Replace(“-“, “+”).Replace(“_”, “/”))
Dim encodedTsAndClientBytes As Byte() = encoding.GetBytes(ts & client)
Dim algorithm As HMACSHA1 = New HMACSHA1(privateKeyBytes)
Dim hash As Byte() = algorithm.ComputeHash(encodedTsAndClientBytes)
Dim signature As String = Convert.ToBase64String(hash).Replace(“+”, “-“).Replace(“/”, “_”)
Return signature
```

C#

```
string ts = DateTime.UtcNow.ToString(“yyyyMMddHHmm”);
string client = “kgua-“ + UserAccessId;
ASCIIEncoding encoding = new ASCIIEncoding();
Byte[] privateKeyBytes = Convert.FromBase64String(password.Replace(“-“, “+”).Replace(“_”, “/”));
Byte[] encodedTsAndClientBytes = encoding.GetBytes(ts + client);
HMACSHA1 algorithm = new HMACSHA1(privateKeyBytes);
Byte[] hash = algorithm.ComputeHash(encodedTsAndClientBytes);
string signature = Convert.ToBase64String(hash).Replace(“+”, “-“).Replace(“/”, “_”);
return signature;
```

A demo application is available to see this code in action (RESTHandler.GetApiSignature()):
http://apikg15doc.kollyapps.com/zip/Demo_API_KOLLYgram.zip

The second field/cryptokey is valid only five minutes after generation.

The production server uses an HTTP Secure connection (HTTPS) which means that information is encrypted again before exiting the client machine (a password is never send in clear view).

Note: the syntax of the headers is important and should be respected.

All elements required to authenticate the user are provided to the customer once their API access has been generated.

Once the authentication system is passed, the current rights of the user are applied.

Note: only the function “Tools/GetUTCServerTime” is consumable without authentication.

Appendix 1

In order to serialize objects that will be sent through the network, JSON (JavaScript Object Notation) standard is used. This is a lightweight text format that supports hierarchy and lots of tools in many programming languages are able to de/serialize from/to JSON.

```json
[
  {
    "productNo": 1,
    "productFrench": "french",
    "productGerman": "german",
    "productEnglish": "english",
    "productItalian": "italian"
  },
  {
    "productNo": 2,
    "productFrench": "french2",
    "productGerman": "german2",
    "productEnglish": "english2",
    "productItalian": "italian2"
  },
  {
    "productNo": 3,
    "productFrench": "french3",
    "productGerman": "german3",
    "productEnglish": "english3",
    "productItalian": "italian3"
  }
]
```

Figure 2 Example of a list of Products with JSON standard

For further details, please refer to the official standard web site: http://json.org/