Ravas
Data Manager
RDM 1.5

V1.5 (dec18)

logitrade logistic systems b.v.

postadres
postbus 12
5328ZG Rossum

bezoekadres
vliegeniersstraat 7b
5405 BH Uden

T +31 413 747 183

E info@logitrade.nl

Rabobank 364326379
IBAN NL89RABO0364326379
SWIFT/BIC-code RABO NL2U

kvk 30266201
btw NL821006332B01
Metaalunie voorwaarden

Contents

Synopsis

RDM or the Ravas Data Manager is a tool that facilitates easy interfacing between any ERP/WMS system and the truck scales from Ravas.
In this way, it is much easier to integrate Ravas truck scales into your logistical operation, RDM allows you to read a weight from a truck at any desired point in the operation.
RDM can be triggered directly from your WMS, or middleware application (e.g. Voice system).

Introduction

In many modern logistical operations the measurement of a weight is desired / required. To name some examples:

- during picking with a truck to check if the correct quantity of items (weight) has been added to the pallet
- weighing a pallet when it is put into a trailer
- etc.

RDM or the Ravas Data Manager is a tool that allows interfacing with the scales integrated into the forklift trucks from Ravas in a modern easy way, without requiring low level knowledge of the scale technology.
Many WMS or ERP systems do not want to perform low level interfacing techniques and don't have the required functions to perform this type of tasks. This is exactly where RDM comes into play.
RDM is a web based service where your WMS can make a request to get the weight reading from one of the forklift trucks deployed in your warehouse.

Requirements

In order to use RDM, following requirements can be stipulated.

- one or more Ravas truck(s) with integrated scale
- Wi-Fi coverage in the area's where wireless communication needs to be established with the truck
- a virtual machine in your data-center or in the cloud to run the RDM service
- modification to one or more of your processes to integrate weighing functionality

In specific situations one or more extra setups might be needed, the name a few:

- VPN tunnel
- additional security measures
- adaptation of middleware that is in use in cooperation with your ERP/WMS system.

RDM briefly explained

Consider the following situation: you have a warehouse, where orders are being picked using a RF terminal connected to the WMS. The warehouse distributes heavy larger parts. Orders are picked using a fork truck. Each order is picked on an individual pallet.
An order is sent to a RF terminal. The picker/truck is directed to a specific warehouse location where one or more parts need to be picked. The weight of each part is known in the WMS. Suppose the part being picked weighs 10 kg. and 3 items are required. At some point the picker will need to confirm the parts are picked and are now lying on the pallet (e.g. by pressing a key). Normally the picker would be directed to go to the next warehouse location, but instead now a step is inserted.
This step will request a weight from this truck. The weight will be compared to what is expected and if the weight is not correct, immediately an extra response can be sent to the RF terminal that the weight is nog correct and the picker needs to add or remove parts. Of course the possible actions here will vary depending on the requirements.

Truck communication

Each truck requires a unique IP address for Wi-Fi communications. The IP address is programmed into the communication controller of the truck.
If you prefer, IP assignment can be done over DHCP, but in this case you need to setup IP assigning based on truck MAC address to allow for a fixed IP address per truck.
This needs to be defined prior to the start of an implementation.

For all communications to take place, this requires some time of course. Your WMS system will make a request to RDM, RDM will in turn communicate with the truck, wait for the reply and give the result back to the WMS. The timing will depend largely on the used hardware, your infrastructure and topology.

On top of this, the electronics of the scale itself sometimes need to wait for a weight to become stable. A maximum wait time of 5 seconds is hard-wired into the scale. In a worst case scenario a request that is made to RDM can take over 5 seconds to complete. For instance if the truck is moving and therefore trembling, a stable weight reading is unlikely. If a request for weighing is done while the truck is moving, the scale will try to get a stable reading, but will most likely fail. In such a case the interface will return an Error after 5 seconds.
Some other specific requests / communications between RDM and the scales might also require a little time to complete. Normally a maximum time of 8 seconds is possible before a reply to a request is gotten. This will only occur in exceptional circumstances as overloading or tilting.

Experiences from the field show that normally if the scale is stable the time required for a cycle will remain well under 100 milliseconds. However if a scale is not stable due to external factors (e.g. movement), then it can take up to 5 seconds before a reply is received.

Interfacing basics

In order for RDM to work together with an ERP, WMS or other sort of back-office system, there needs be setup some interface. In other words some sort of coupling between the two systems. Interfacing is very common in the field of Information Technology and most likely you have heard of CSV file exchange, which is also a form of interface that can be setup, but it is kind of old.

Since CSV other interface techniques have evolved, like Edifact, XML, SOAP, Rest. The last one, Restful interfacing using JSON format has lately become the preferred choice for web based interfacing.

A RESTful web service is based on representational state transfer (REST) technology, an architectural style and approach to communications often used in web services development.
More specific, a RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.

With Rest interfacing there always needs to be one side that acts like a (web)server and one side that acts like a client.
A. The server facilitates a point where requests can be made.
B. The client performs requests towards the server.
For the RDM REST interface only POST and GET are used and RDM acts as the Rest server.

What RDM API does?

The RDM Rest API allows developers to make requests to the RDM service. As explained below 7 types of requests are possible. Each request has its own response type. This will be detailed below.

The API is a Rest full web service which allows easy integrations within other development platforms such as PHP, Java C#.net etc.

Connect to the API

To connect to the RDM Rest API web services a URL will need to be setup for your environment. This will need to be done by your IT department.

Authentication

The API uses HTTP Basic Auth for authentication over SSL (https://). You will need to send an access token as the HTTP username for each API request.
Please contact Ravas/Logitrade to obtain a valid access token to access this API, normally this will be discussed as part of the implementation.

RDM functions available

Following functions are available through Rest:

1. getTrucks
2. getTruckStatus
3. initTruck
4. tareTruck
5. getGrossWeight
6. getNetWeight
7. checkWeightOnTruck (under construction)

A function is called by doing a GET/POST request to a specific url. Let's assume you have the RDM service running on a server that is reachable at rdm.<client-domain>.com, then a GET might look something like:

GET https://rdm-api.<client-domain>.com/rdm/v1/getTrucks

RDM – getTrucks

This function will return all fork trucks that are setup in your RDM. If no trucks are setup you will receive an error message as string.

Normally you will receive an array with truck details with following information per truck:

- Truck Uid (unique identifier for RDM; e.g. a barcode assigned to the truck)
- Truck type (normally ravas indicator)
- Truck name (description)
- Truck ip

RDM – getTruckStatus

This function requires you to add a Truck identifier to the request, like:

GET https://rdm-api.<client-domain>.com/rdm/v1/getTruckStatus/<TruckUId>

This function can return an error message in case the truck is not known or in case the truck doesn't respond.

On success the function will return a data array giving some details on the truck:

- Truck Uid (unique id for RDM; e.g. a barcode assigned to the truck)
- Truck type
- Truck name
- Truck ip
- Last measurement
- Weighing cell status
- Tare active
- Max load exceeded

RDM – initTruck

This function requires you to add a Truck identifier to the request, like:

POST https://rdm-api.<client-domain>.com/rdm/v1/initTruck/<TruckUId>

This function can return an error message (ERR) in case the truck is not known or in case the truck doesn't respond.

This command prepares a truck for weighing operations and defines the current state of the truck as the 'zero-position'. This command is normally issued before a carrier (e.g. empty pallet) is added to the truck to commence operations.

On success the function will return OK.

RDM – tareTruck

This function requires you to add a Truck identifier to the request, like:

POST https://rdm-api.<client-domain>.com/rdm/v1/tareTruck/<TruckUid>

This function performs a tare on the weighing scale as to set the current load as the new zero value. Doing tares keeps the scale in the optimal weighing range (0-200kg). A tare operation can be done multiple times, normally after every pick operation when goods are added to the pallet, a new tare is done. A subsequent tare cancels the previous one and sets a new tare value which includes the old tare value and the added net weight. If the weight doesn't get stable within 5 seconds an error will be generated.

This function can return an error message in case the truck is not known or in case the truck doesn't respond or takes too long to perform the tare.

On success the function will just return: 'Ok'.

RDM – getGrossWeight

This function requires you to add a Truck identifier to the request, like:

GET https://rdm-api.<client-domain>.com/rdm/v1/getGrossWeight/<TruckUid>

This function can return an error message in case the truck is not known or in case the truck doesn't respond or in case the weight obtained is not stable.
If the truck is in rest, this function will return a stable weight reading normally within 100 msec. however it can take up to 5 seconds for this function to respond in case the truck is in motion or in case items are being loaded onto the pallet.

On success the function will return a value, being the weight in kg. This will be a float value. This is the Gross weight and ignores any tares in place. It just gives the actual total gross weight.

RDM – getNetWeight

This function requires you to add a Truck identifier to the request, like:

GET https://rdm-api.<client-domain>.com/rdm/v1/getNetWeight/<TruckUid>

This function can return an error message in case the truck is not known or in case the truck doesn't respond or in case the weight obtained is not stable.
If the truck is in rest, this function will return a stable weight reading normally within 100 msec. however it can take up to 5 seconds for this function to respond in case the truck is in motion or in case items are being loaded onto the pallet.

On success the function will return a value, being the weight in kg. This will be a float value. This is the effective Net weight since the previous last Tare.

Example:
Suppose an empty truck is loaded with an empty pallet of 15 kg. When the pallet is loaded a Tare function is sent to the truck. After this the pallet is loaded with a crate with a weight of 10 kg.
In this case the Net weight function will return 10 kg and Gross weight will return 25 kg.

RDM – checkWeightOnTruck

This function requires you to add a Truck identifier to the request, like:

POST https://rdm-api.<client-domain>.com/rdm/v1/checkWeightOnTruck/<TruckUid>

This function can return an error message in case the truck is not known or in case the truck doesn't respond or in case the weight obtained is not stable.

This function will be further detailed in a next release of this document.

API responses

The Rest service support JSON and XML formats for input and output data by default. The default format is JSON. Use the Accept Header (application/xml or application/json) to negotiate between JSON and XML.
When an XML String is sent to a REST service, you must also set Content-Type HTTP header to be "application/xml".
By default the rest Api will respond with the appropriate HTTP status code to indicate the results of an action. See a list of HTTP status codes below:

- 200: OK. Everything worked as expected.
- 201: A resource was successfully created in response to a POST request. The Location header contains the URL pointing to the newly created resource.
- 204: The request was handled successfully and the response contains no body content (like a DELETE request).
- 304: The resource was not modified. You can use the cached version.
- 400: Bad request. This could be caused by various actions by the user, such as providing invalid JSON data in the request body, providing invalid action parameters, etc.
- 401: Authentication failed.
- 403: The authenticated user is not allowed to access the specified API endpoint.
- 404: The requested resource does not exist.
- 405: Method not allowed. Please check the Allow header for the allowed HTTP methods.
- 415: Unsupported media type. The requested content type or version number is invalid.
- 422: Data validation failed (in response to a POST request, for example). Please check the response body for detailed error messages.
- 429: Too many requests. The request was rejected due to rate limiting.
- 500: Internal server error. This could be caused by internal program errors.

In case of an error an additional message will be send in the response body.

Example:

```
HTTP/1.1 404 Not Found
Date: Sun, 02 Mar 2014 05:31:43 GMT
Server: Apache/2.2.26 (Unix) DAV/2 PHP/5.4.20 mod_ssl/2.2.26 OpenSSL/0.9.8y
Transfer-Encoding: chunked
Content-Type: application/json; charset=UTF-8
{
"name": "Not Found Exception",
 "message": "The requested resource was not found.",
 "code": 0,
 "status": 404
}
```

Instead of relying on using different HTTP statuses to indicate different errors it is also possible to always return the 200 as HTTP status and enclose the actual HTTP status code as part of the JSON/XML structure in the response.

To enable this behavior you can add an optional URL GET parameter to each Api request:

```
suppress_response_code=true
```

More information needed?

In case you need more information or you need to discuss things in detail, please make a request to your contact person at Ravas.

RDM Function details

getTrucks

Request a list of trucks setup in your RDM.

Resource:
/v1/getTrucks

Method:
GET

Data format:
JSON/XML

URL Params:
none

Data params:
none

Success response:
When successful, this method will return all trucks setup in this instance of RDM.

Example response:

```
{
   "Trucks":    [
            {
         "TruckUid": "*A*",
         "TruckType": "RavasIndicator",
          "TruckName": "Ravas1",
         "TruckIp": "192.168.1.31",
      },
            {
         "TruckUid": "*B*",
         "TruckType": "RavasIndicator",
         "TruckName": "Ravas2",
         "TruckIp": "192.168.1.33",
      }
   ]
}
```

Error response:

When not successful, this method will return an error message. See also chapter "Api responses"
Example error response:

```
{
   "name": "Unprocessable entity",
   "message": "No trucks defined",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
```

getTruckStatus

Request the status of a specific truck setup in your RDM.

Resource:
/v1/getTruckStatus

Method:
GET

Data format:
JSON/XML

URL Params:
truck_id

Data params:
none

Success response:
When successful, this method will return the status of the specified truck.

Example response:


```
{
   "TruckUid": "*A*",
   "TruckType": "RavasIndicator",
   "TruckName": "Ravas1",
   "TruckIp": "192.168.1.31",
   "LastMeasurement": "2018-07-15 22:53:03",
   "WeighingCellStatus": "OK",
   "TareActive": "YES",
   "MaxLoadExceeded": "NO",
}
```


Error response:
When not successful, this method will return an error message. See also chapter "Api responses"
Example error responses:


```
{
   "name": "Unprocessable entity",
   "message": "Unknown TruckUId: 99",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
or
{
   "name": "Unprocessable entity",
   "message": "Socket connect error: 111-Truck already busy communicating",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
```

initTruck

Initializes a specific truck in your environment for operations. I.e. the scale is checked and the scale is zeroed.

Resource:
/v1/initTruck

Method:
POST

Data format:
JSON/XML

URL Params:
truck_id

Data params:
none

Success response:
When successful, this method will OK.

Example responses:

```
{
   "TruckUid": "*A*",
   "Init": "OK",
}
```

Error response:
When not successful, this method will return an error message. See also chapter "Api responses"
Example error response:

```
{
   "name": "Unprocessable entity",
   "message": "Ravas Indicator returned ERR while initializing",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
or
{
   "name": "Unprocessable entity",
   "message": "Socket connect error: 110-Connection timed out",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
```

Note: this error can occur if a truck has been switched off and is not communicating.

```
or
{
   "name": "Unprocessable entity",
   "message": "Socket connect error: 113-No route to host",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
```

Note: this error can occur if invalid ip nummer is setup in RDM

tareTruck

Request the status of a specific truck setup in your RDM.

Resource:
/v1/tareTruck

Method:
POST

Data format:
JSON/XML

URL Params:
truck_id

Data params:
none

Success response:
When successful, this method will return the new order including all details.

Example response:

```
{
   "TruckUid": "*A*",
   "Tare": "OK",
}
```

Error response:
When not successful, this method will return an error message. See also chapter “Api responses”
Example error response:

```
{
   "name": "Unprocessable entity",
   "message": "Error while taring, scale unstable",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
```

getGrossWeight

Request a stable weight reading from a specific truck.

Resource:
/v1/getGrossWeight

Method:
GET

Data format:
JSON/XML

URL Params:
truck_id

Data params:
none

Example response:

```
{
   "TruckUid": "*AA*",
   "Weight": 7.73,
   "WeightUnit": "kg",
}
```

Error response:
When not successful, this method will return an error message. See also chapter "Api responses"
Example error response:

```
{
   "name": "Unprocessable entity",
   "message": "No stable weight",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
```

getNetWeight

Request a stable weight reading from a specific truck.

Resource:
/v1/getNetWeight

Method:
GET

Data format:
JSON/XML

URL Params:
truck_id

Data params:
none

Example response:

```
{
   "TruckUid": "*A*",
   "Weight": 7.73,
   "WeightUnit": "kg",
}
```

Error response:
When not successful, this method will return an error message. See also chapter "Api responses"
Example error response:

```
{
   "name": "Unprocessable entity",
   "message": "No stable weight",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
```

checkWeightOnTruck

Request the status of a specific truck setup in your RDM.

Resource:
/v1/checkWeightOnTruck

Method:
POST

Data format:
JSON/XML

URL Params:
truck_id

Data params:
Order Object:

```
{
        "TruckUid": "*A*",
        "WeightToCheck": 125,
}
```

```
Under development.
```

Success response:
Under construction

Example response:

```
{
}
```

Error response:
When not successful, this method will return an error message. See also chapter "Api responses"
Example error response:

```
{
   "name": "Unprocessable entity",
   "message": "...",
   "code": 0,
   "status": 422,
   "type": "yii\\web\\UnprocessableEntityHttpException"
}
```

Error conditions from Truck

This table shows possible error conditions that might be returned from a truck.

01	LOAD CELL SIGNAL UNSTABLE	
02	IFORKS OVERLOADED ON MAXIMUM CAPACITY	
03	TARA WHILE NEGATIVE WEIGHT	
04	ZERO OUT OF RANGE	
06	IFORKS OVERFLOW ADC	
08	CALIBRATION OUT OF RANGE NEGATIVE	
09	CALIBRATION OUT OF RANGE SIGNAL TOO LOW	
10	CALIBRATION POINT LOWER THAN PREVIOUS POINT	
21	COMMUNICATION FAILURE FORK 1	
22	COMMUNICATION FAILURE FORK 2	
23	COMMUNICATION FORK 1 TOO FEW SAMPLES received	
24	COMMUNICATION FORK 2 TOO FEW SAMPLES received	
25	COMMUNICATION FAILURE 1AD	
26	COMMUNICATION 1AD TOO FEW SAMPLES received	
40	LEVEL MAX	
41	OIML restriction while printing	
42	NTEP restriction while printing	
43	OIML restriction while calibration	
44	NTEP restriction while calibration	
45	CALIBRATION NOT ALLOWED PROTECTED BY JUMPER	
46	AUDITTRAIL OUT OF RANGE	
60	LOW BAT INDICATOR	
61	LOW BAT FORK 1	
62	LOW BAT FORK 2	
71	OFF CENTRE LOAD TIP (only active when P13 ≠ no)	
72	OFF CENTRE LOAD SIDE (only active when P13 ≠ no)	
80	ERROR in RDC transfer	
81	RDC buffer full	
92	GROSS NEGATIVE UNDERLOAD	
98	CALIBRATION POINT MUST BE HIGHER THAN PREVIOUS ONE	
99	ZEROING WHILE UNIT SWITCHED	