Web Services for E-Commerce Reference

Version: 1.0r6

Cleveron, info@cleveron.eu

April 16, 2015

Introduction

This is a reference of Web services e-commerce systems will use to integrate with the parcel delivery network. The interface (API) supports three primary use cases needed by e-commerce systems:

1. Getting the list of available destinations for displaying a choice of delivery locations to the customers at checkout.
2. Creating delivery orders for delivering parcels to the customers.
3. Tracking the status of parcels.

The interface follows loosely the principles of REST [2] and uses HTTP over TLS [1] as the secure transport mechanism. Supported data formats are JSON and XML; UTF-8 encoding is used by default throughout the interface.

1 Resources

The following is a list of resources accessible through the interface. E-commerce systems are integrated with the parcel delivery network by querying and updating these resources.

- `places` — available destinations (Automated Parcel Terminals, APTs)
- `orders` — parcel delivery orders
- `traces` — parcel tracking information
- `languages` — supported language tags

The representations of the resources and the requests supported on these resources are specified in the following subsections along with usage examples.

Each resource consists of a number of named fields which are listed in the descriptions of the resources. The fields have types given in square brackets after the name of the field. Type information includes the maximal length of the value in case of string values and the precision in case of decimals.

1.1 Resource `places`

This is a read-only resource for querying the list of available destination APTs for displaying delivery options to the customer at checkout. An e-commerce system should update the list of locations periodically, e.g., daily or weekly, and store the data locally in its database.

- `place_id` [integer] — a unique identifier of the APT; used for specifying the destination of a delivery order.
- `name` [string(70)] — name of the APT
- `address` [string(100)] — street address of the APT location
- `group` [string(35)] — group name used for organising the list, e.g., the province or region of the APT location.

```
[
  {
    "place_id":1001,
    "name":"Extra Nice Shopping Centre",
    "address":"Street 123, City",
    "open":"Mon-Fri from 9am to 8pm"
  },
  {
    "place_id":2007,
    "name":"The Gas Station",
    "address":"1001 Another Street, Town",
    "open":"From 9am to 8pm, except holidays"
  },
  ...
]
```

Figure 1.1: Example list of APTs in JSON format

- `subgroup` [string(35)] — subgroup name used for organising the list, e.g., the city or town of the APT location.
- `open` [string(75)] — opening hours, e.g.: *Mon-Fri from 9am to 8pm.*
- `findme` [string(254)] — comment helping to find the APT, e.g.: *Next to the information desk.*
- `geolat` [decimal(8,6)] — geographic coordinates: latitude
- `geolng` [decimal(9,6)] — geographic coordinates: longitude

The fields `name`, `open` and `findme` of this resource are localized. See Subsection 1.5 for details about localization. The fields `place_id` and `name` have always non-empty values, other fields may be empty. The purpose of the fields `group` and `subgroup` is helping to organise a long list of destinations hierarchically in a user-friendly way.

1.1.1 Examples

Figures 1.1 and 1.2 show a sample list of APTs returned as a response body to a GET request to the `places` resource, correspondingly in JSON and XML formats.

1.2 Resource `orders`

This is the resource used for creating new delivery orders and getting the information of existing orders. This resource supports several methods. The `POST` method is used for creating new orders, `GET` is used for listing existing orders and fetching order data, `PATCH` and `PUT` are used for updating or replacing an order, while `DELETE` can be used for cancelling an order. See Section 5 for more details on the semantics of supported HTTP methods.

```
<places>
  <place href="https://host/path/places/1001">
    <place_id>1001</place_id>
    <name>Extra Nice Shopping Centre</name>
    <address>Street 123, City</address>
    <open>Mon-Fri from 9am to 8pm</open>
  </place>
  <place href="https://host/path/places/2007">
    <place_id>2007</place_id>
    <name>The Gas Station</name>
    <address>1001 Another Street, Town</address>
    <open>From 9am to 8pm, except holidays</open>
  </place>
  ...
</places>
```

Figure 1.2: Example list of APTs in XML format

The following is the list of the fields of an order. Note the mandatory, optional and read-only fields. Values of the mandatory fields have to be given in `POST` requests that are used for creating new orders. Read-only fields contain values generated by the system automatically and attempts to change these values will result in an error.

- `barcode` [string(100)] — barcode generated by the system when a new order is created, used for tracking orders; ASCII alphanumeric (read-only)
- `barcode_alt` [string(100)] — an optional third party barcode which can be used for handling parcels with third party labels; ASCII alphanumeric, underscore
- `place_id` [integer] — destination APT identifier from the list in the `places` resource (mandatory)
- `name` [string(254)] — recipient name (mandatory)
- `phone` [string(50)] — recipient phone number for SMS delivery (mandatory)
- `email` [string(254)] — recipient e-mail address (mandatory)
- `content` [string(254)] — optional description of content
- `comment` [string(1000)] — optional comment for, e.g., assisting in packaging or manual labelling
- `refcode` [string(254)] — optional customer's reference to this order, can be used for querying order data and for connecting the order to records in the customer's database.
- `language` [string(25)] — language preference for communication with the recipient (SMS, e-mail); from the `languages` resource.

- `cod_amount` [integer] — Cash On Delivery amount that the recipient has to pay before receiving the parcel. The amount is represented in *minor* units of the currency specified in `cod_currency`. Examples:
  - EUR 123.45 → `cod_amount = 12345 cod_currency = EUR`
  - CLP 12345 → `cod_amount = 12345 cod_currency = CLP`

  Values in the range $[0, 2e9]$ are accepted. The value cannot be updated after creating an order.
- `cod_amount_dec` [decimal] — Cash On Delivery amount that the recipient has to pay before receiving the parcel. The amount is represented in *major* units of the currency specified in `cod_currency`. The value may have an optional fractional part separated by a decimal point (“.”). Examples:
  - EUR 123.45 → `cod_amount_dec = 123.45 cod_currency = EUR`
  - CLP 12345 → `cod_amount_dec = 12345 cod_currency = CLP`
- `cod_currency` [string(3)] — ISO 4217 [5] currency code (mandatory when `cod_amount` is specified, cannot be updated)
- `timestamp` [string] — order creation date in RFC3339 [6] format (read-only)
- `status` [string] — status of the order, values from Table 1.1 (read-only)
- trace [string] — URI of the trace of this order (read-only)

Note that when creating an order the fields `cod_amount` and `cod_amount_dec` are mutually exclusive and specifying both is an error even when the values match. When requesting COD order information, both fields are returned in responses. See Figure 1.7 for an example. It is also an error to specify a value of the `cod_amount_dec` field that cannot be represented in minor units on the `cod_amount` field without loss of precision. For example: `cod_currency = EUR cod_amount_dec = 12.345` is rejected because rounding would cause loss of precision.

This resource supports filtering by `barcode_alt` field. See Section 5.1.1 for filtering details.

1.3 Examples

To create an order, a POST request has to be sent to the `orders` resource. Figure 1.3 shows an example of request data. On successful request the data show in Figure 1.4 is returned. Equivalent requests in JSON format are show in Figures 1.5 and 1.6.

1.4 Resource `traces`

This is a read-only resource used for tracking parcels and the status of corresponding delivery orders. A trace of a parcel consists of a sequence of events with timestamps. Each event also includes the new status of the order. Optionally, a location and an initiator related to the event may be included where applicable. The following is the list of fields of an event record:

Status tag	Description
`new`	New order
`to-term`	Parcel is on the way to logistics terminal
`in-term`	Parcel is in logistics terminal
`to-apt`	Parcel is on the way to APT
`in-apt`	Parcel is in APT
`delivered`	Parcel is delivered to the recipient
`ret-in-apt`	Returned parcel is in APT
`ret-to-term`	Returned parcel is on the way to logistics terminal
`ret-in-term`	Returned parcel is in logistics terminal
`ret-to-apt`	Returned parcel is on the way to APT
`ret-delivered`	Returned parcel is delivered to the customer
`lost`	Parcel is lost
`cancelled`	Order was cancelled

Table 1.1: Order status values

```
<order>
  <place_id>2007</place_id>
  <name>John Smith</name>
  <phone>+123451234</phone>
  <email>john@smith.com</email>
</order>
```

Figure 1.3: Contents of a request to create a delivery order in XML

```
<order href="https://host/path/orders/EE123456789">
  <barcode>EE123456789</barcode>
  <place_id>2007</place_id>
  <name>John Smith</name>
  <phone>+123451234</phone>
  <email>john@smith.com</email>
  <language>en-GB</language>
  <timestamp>2012-06-21T08:50:08+03:00</timestamp>
  <status>new</status>
  <trace href="https://host/path/traces/EE123456789"/>
</order>
```

Figure 1.4: Response to an order creation request in XML

```
{
  "place_id": 2007,
  "name": "John Smith",
  "phone": "+123451234",
  "email": "john@smith.com"
}
```

Figure 1.5: Contents of a request to create a delivery order in JSON

```
{
  "barcode": "EE123456789",
  "place_id": 2007,
  "name": "John Smith",
  "phone": "+123451234",
  "email": "john@smith.com",
  "language": "en-GB",
  "timestamp": "2012-06-21T08:50:08+03:00",
  "status": "new",
  "trace": { "href": "https://host/path/traces/EE123456789" }
}
```

Figure 1.6: Response to an order creation request in JSON

```
<order href="https://host/path/orders/EE123456789">
  <barcode>EE123456789</barcode>
  <place_id>2007</place_id>
  <name>John Smith</name>
  <phone>+123451234</phone>
  <email>john@smith.com</email>
  <language>en-GB</language>
  <timestamp>2012-06-21T08:50:08+03:00</timestamp>
  <status>new</status>
  <cod_currency>EUR</cod_currency>
  <cod_amount>123456</cod_currency>
  <cod_amount_dec>1234.56</cod_currency_dec>
  <trace href="https://host/path/traces/EE123456789"/>
</order>
```

Figure 1.7: Response to a COD order creation request in XML

- `type` [string(35)] — a tag specifying the type of an event (e.g.: `create`, `cancel`, `delivery`)
- `timestamp` [string] — time stamp of the event in RFC 3339 [6] format
- `status` [string(35)] — status of the delivery order after the event; see Subsection 1.2 and Table 1.1 for details.
- `location` [string(100)] — the name of the location where the event occurred
- `initiator` [string(100)] — the name of the initiator of the event

Without specifying a subresource, the `traces` resource outputs a list of available traces. The barcode of an order has to be specified as a subresouce to get the details of a trace, i.e., the request has to be directed to an address in the following form: `https://host/path/traces/EE123456789`.

1.4.1 Examples

A `GET` request of the `traces` resource lists available traces as shown in Figures 1.8 and 1.9. The details of traces can be examined by following the links included in the response or by specifying a valid barcode as a subresource in the URI. Example representations of a specific trace are shown in Figures 1.10 and 1.11.

1.5 Resource `languages`

This is a read-only resource containing the list of supported language tags. A language tag consists of two parts separated by a dash (minus) character. The first part is a two-letter language code from ISO 639-1 [4] specification. The second part is a two-letter country code from ISO 3166-1 [3]. The following is a sample list of language tags along with their meanings that might be returned on a `GET` request to the `languages` resource. The resource also specifies a default language tag.

```
<traces>
  <trace barcode="EE123456789"
         href="https://host/path/traces/EE123456789"/>
  <trace barcode="ES1234567890"
         href="https://host/path/traces/ES1234567890"/>
  ...
</traces>
```

Figure 1.8: List of traces in XML

```
[
  { "href": "https://host/path/traces/EE123456789",
    "barcode": "EE123456789" },
  { "href": "https://host/path/traces/ES1234567890",
    "barcode": "ES1234567890" },
    ...
]
```

Figure 1.9: List of traces in JSON

```
<trace barcode="EE123456789">
  <event type="created" timestamp="2012-06-21T08:50:08+03:00"
    status="new"/>
  <event type="cancelled" timestamp="2012-06-21T09:50:08+03:00"
    status="cancelled" initiator="Customer X"/>
</trace>
```

Figure 1.10: Representation of a trace in XML

```
[
  { "type": "created",
    "timestamp": "2012-06-21T08:50:08+03:00",
    "status": "new" },
  { "type": "cancelled",
    "timestamp": "2012-06-21T09:50:08+03:00",
    "status": "cancelled",
    "initiator": "Customer X" }
]
```

Figure 1.11: Representation of a trace in JSON

- `es-ES` — Spanish
- `eu-ES` — Basque
- `ca-ES` — Catalan
- `en-GB` — British English

Language tags are used for interface localization. When the client expects a localized answer, the `Accept-Language` header has to be set accordingly. Currently, the only resource supporting localization is `places`. When no `Accept-Language` header is specified, the default language is assumed. The server includes the `Content-Language` in its responses. If an unsupported language is requested, the server will reply with a status code of 406 (not acceptable). Language ranges are not supported.

1.5.1 Examples

Example representations of the `languages` resource are shown in Figures 1.12 and 1.13. Although the language tags may be represented in mixed case letters, the values used in HTTP headers are case insensitive.

2 Transport Security

Data exchange is performed over an encrypted channel using HTTPS protocol. Clients are encouraged to confirm the server's identity by verifying its certificate at the beginning of each session. The details of the server address and certificate fingerprints are distributed to clients separately from this document.

3 Authentication

All requests have to be authenticated. The only authentication mechanism available is HTTP Basic authentication which is widely supported by HTTP clients and libraries. Usually, the authentication details are handled by clients automatically. For example, to use authentication with the `curl` command line program:

```
curl -u username:password https://host/path/
```

```
<languages>
  <language default="true" tag="es-ES"/>
  <language tag="eu-ES"/>
  <language tag="ca-ES"/>
  <language tag="en-GB"/>
</languages>
```

Figure 1.12: Representation of languages resource in XML

```
[
  { "tag": "es-ES",
    "default": "true" },
  { "tag": "eu-ES" },
  { "tag": "ca-ES" },
  { "tag": "en-GB" }
]
```

Figure 1.13: Representation of languages resource in JSON

At the protocol level, each request has to include the `Authorizaton` header. The header should contain a value `Basic` x, where x is obtained by Base64-encoding the string *username*`:`*password*. For example, when user name is `username` and password is `password`:

```
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

4 Request and Representation Formats

All resources can be represented in different formats, in particular, the formats supported in current specification version are JSON and XML. It is also acceptable for a client to send request data in one format, e.g., XML, and to notify the server to give the response in another format, e.g., JSON. The formats are specified using standard HTTP headers `Accept` and `Content-Type`. The following are the supported content types:

- `application/vnd.cleveron+json; version=1.0`
- `application/vnd.cleveron+xml; version=1.0`

By default, when no `Accept` or `Content-Type` header is specified for requests, JSON format is assumed for both. The `version` parameter will be used to support future extensions and backwards compatibility. The version number *will not* be incremented for minor/backwards compatible changes. When no version is specified, the latest supported version is assumed. Note that the current implementation does not support HTTP media ranges.

For example, when a client sends a request in XML and expects the response in JSON, the client would set the following request headers:

```
Accept: application/vnd.cleveron+json
Content-Type: application/vnd.cleveron+xml
```

As JSON is the default format, the first header can be omitted in this case. In addition, to save bandwidth and reduce response times, clients may request compression by setting the header `Accept-Encoding`, for example:

```
Accept-Encoding: gzip, deflate
```

In that case the server might respond with a compressed response body.

5 Resource Methods

Five standard HTTP methods are used: `GET`, `POST`, `PUT`, `PATCH` and `DELETE`. The semantics of these methods in the context of this interface is discussed below.

5.1 Method `GET`

The `GET` method can be used for retrieving a representation of a resource. For example:

```
curl -X GET -u username:password \
  -H "Accept: application/vnd.cleveron+xml; version=1.0" \
  -H "Content-Type: application/vnd.cleveron+xml; version=1.0" \
  https://host/path/
```

`GET` methods are guaranteed to not modify any resources, the requests may be repeated and the responses may be cached by the client or a proxy.

5.1.1 Filtering in `GET` requests

Filters can be specified in HTTP query string:

```
curl -X GET -u username:password \
  -H "Accept: application/vnd.cleveron+xml; version=1.0" \
  -H "Content-Type: application/vnd.cleveron+xml; version=1.0" \
  https://host/path/orders?barcode_alt=ABC
```

This query returns a list of orders where the field `barcode_alt` is equal to "ABC". If no such orders exist, an empty list is returned.

5.2 Method `POST`

The `POST` method is used for creating new resources. The only resource that accepts `POST` requests is `orders`. The response contains a representation of the created resource.

5.3 Method `PUT`

The `PUT` method is used to completely replace an existing resource. This method can be applied on the `orders` resource while the status of the target order is `NEW`. The response contains a representation of the new contents of the resource.

5.4 Method `DELETE`

The `DELETE` method is used to cancel an order. Orders cannot be cancelled after the delivery to the customer. The response body is empty. The effect of the request is irreversible, i.e., the cancellation cannot be undone.

5.5 Method PATCH

The PATCH method is used for updating selected fields of a resource. The only resource supporting this method is orders. This method can be used for redirecting a not yet delivered parcel to an alternate destination or specify the value of a third party barcode. The response contains a representation of the updated contents of the resource.

6 Responses

All responses include an HTTP response code. The following is a list of possible codes and their meanings.

- 200: Success (upon a successful GET, PUT, or DELETE request)
- 201: Created (upon a successful POST request)
- 400: Resource Invalid (improperly formatted request)
- 401: Unauthorized (incorrect or missing authentication credentials)
- 404: Resource Not Found
- 405: Method Not Allowed
- 406: Not Acceptable
- 500: Application Error

Successful POST requests return a status code of 201, include a Location header containing the URI of the created resource, and include a representation of the resource in the body of the response. Successful PUT requests return a status code of 200, and include a representation of the updated resource in the body of the response. Successful DELETE requests return a status code of 200, and an empty body.

7 Revision History

Table 7.1 gives an overview of specification versions and significant changes between the revisions.

Version	Date	Description
1.0r2	May 23, 2014	Initial public version
1.0r3	November 12, 2014	Added geolat, geolng to places; fixed typos
1.0r4	January 8, 2015	Added cod_amount, cod_currency
1.0r5	March 19, 2015	Added order filtering by barcode_alt field
1.0r6	April 16, 2015	Added cod_amount_dec, allow underscore in barcode_alt

Table 7.1: Revision history and changes

References

[1] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. Updated by RFCs 5746, 5878, 6176.

[2] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.

[3] International Organization for Standardization. *ISO 3166-1:2006 Codes for the representation of names of countries and their subdivisions — Part 1: Country codes*. ISO, Geneva, 2 edition, 2006.

[4] International Organization for Standardization. *ISO 639-1:2002 Codes for the representation of names of languages — Part 1: Alpha-2 code*, 2002.

[5] International Organization for Standardization. *ISO 4217:2008 – International Standard for currency codes*, 2008.

[6] G. Klyne and C. Newman. Date and Time on the Internet: Timestamps. RFC 3339 (Proposed Standard), July 2002.