Clang Data Model REST API

User manual

Table of Contents

1. Introduction

Clang Extended Data allows you to define a model for data that can be related to customers present in the Clang CRM. For more information on Clang Extended Data please refer to the appropiate documents. This manual describes how to access the data that are stored as Clang Extended Data by way of the public REST API (Application Programming Interface). This interface allows the following operations on the data contained in a data model:

- reading
- inserting
- updating
- deleting

As an added feature, the REST API is self-documenting, in that you may request meta-information on the actual data model using the REST API itself.

2. Requirements

In order to use the REST API you need the following at your disposal:

- Any programming environment that allows you to make HTTP requests and process the responses will allow you to make REST API calls. In particular, you need to be able to generate and parse JSON (JavaScript Object Notation).
- You need a base URL and an access token, that you may obtain by requesting these at E-Village. Note that if you already have a SOAP token for the brand you wish to access, you may also use this with the REST API.

3. Data model

This manual uses a simple data model as the basis for explaining how to use the REST API. Suppose we're tracking orders at a restaurant that delivers pizzas. A customer may have placed any number of orders, and each order may contain any number of pizzas. The choice of pizzas however is from a menu which is the same for all customers. This is expressed in the data model displayed in Illustration 1.

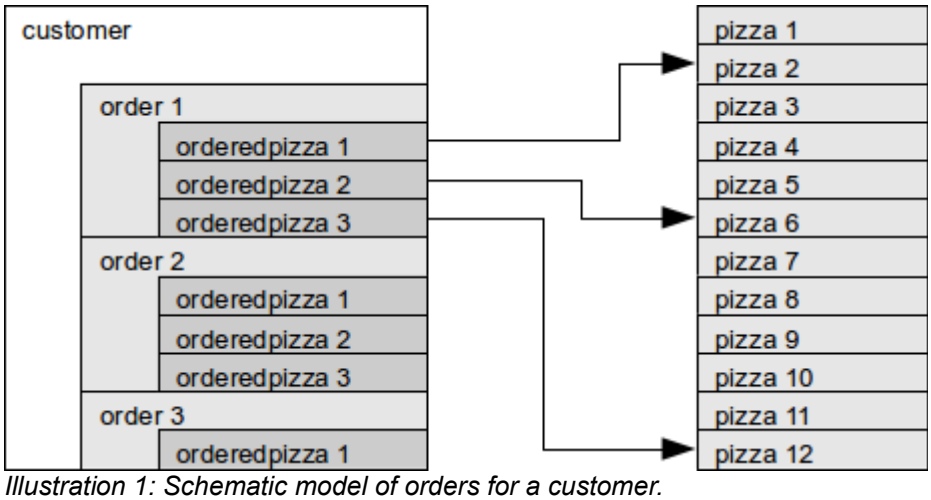

Illustration 1: Schematic model of orders for a customer.

The depicted customer has placed three orders over time, the first two orders containing three different types of pizza, and the third order only one. The ordered pizzas refer to a list that contains information on the pizzas themselves. In the schematic these references are only displayed for the first order. Each customer in the Clang CRM has a record with orders, but these all refer to the same list of pizzas.

The design described above is expressed in the model displayed in Illustration 2.

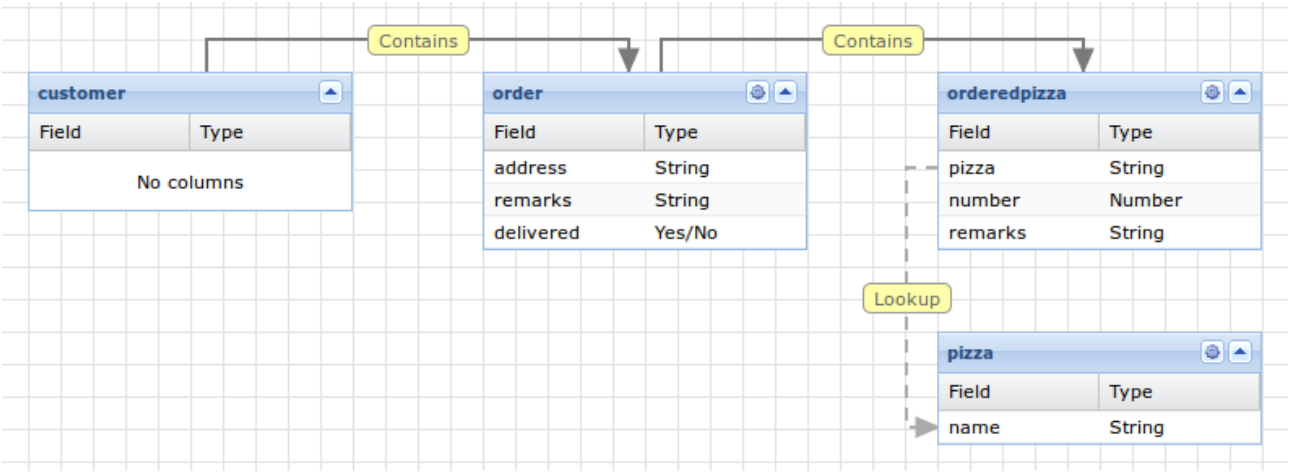

Illustration 2: Data model of the pizza orders.

The model has the following properties:

- The customer table is always defined, and has no columns, since the customer data are stored in the Clang CRM.
- A customer *contains* one or more orders. In this simple model an order has a delivery address, a text field for remarks, and a status flag indicating whether or not the order has been delivered.
- An order *contains* one or more ordered pizzas. An ordered pizza stores the type of pizza, the number of pizzas ordered for this type, and another text field for remarks.
- An ordered pizza *looks up* additional information on the pizza by means of the pizza column, which actually stores a reference to the pizza table.

4. REST requests

4.1. Basics

The basic concept behind REST (REpresentational State Transfer) is twofold:

- A resource is uniquely identified by a URL.
- Operations are implemented as HTTP requests using a specific method.

By making correctly composed requests to a specific URL using the correct HTTP request method, all operations provided by the API can be performed.

4.1.1. REST URLs

The base URL for the Clang Data Model REST API is:

```
https://example.com/app/api/rest/public/v2/dataextension
```

The domain name depends on the portal and brand you intend to make requests for. Addressing resources in the data model is done by appending the resource names and when necessary the resource ids to the URL. For example, the URL

```
https://.../dataextension/pizzas
```

accesses all resources in the *pizza* table. In this example and all examples below the URL is displayed shortened, but of course all requests need to be done using the complete URL. The URL

```
https://.../dataextension/pizzas/clang_523ae306ee3bf
```

identifies only the pizza with id *clang_523ae306ee3bf* in the *pizzas* table. All resources in tables in the data model automatically receive an id field named *clang_id* with a generated unique content with the prefix '*clang_*'. It is not possible to assign ids using the API or any other way. A special id is used for the *customer* table. Customers are assigned the Clang CRM customer id, prefixed with '*clang_*'. As an example, the record for the Clang customer with customer id *42* is addressed as

```
https://.../dataextension/customer/clang_42
```

When tables are related using a *contains* relation, they are addressed as subresources within the resource that contains them. For example

```
https://.../dataextension/customer/clang_42/order
```

accesses all orders by the customer with id *42*. Taking it one step deeper

```
https://.../dataextension/customer/clang_42/order/clang_523ae3385d12c/orderedpizza
```

points to all the pizzas in the order with id *clang_523ae3385d12c* for the customer with id *42*. When a subresource is requested, the URL must identify all containing resources. For example, it is not possible to access order data without specifying the customer. The following URL will return an error:

```
https://.../dataextension/customer/order/
```

NB: the *customer* table is always defined. Regardless of the locale you may be working in in Clang, the REST API only recognizes the name '*customer*'.

4.1.2. REST methods

The REST API supports the following methods. The results of the request may vary depending on whether you specify an id in the resource URL.

HTTP method	without id	with id
GET	return all resources	return identified resource
POST	add new resource	invalid request
PUT	invalid request	update identified resource
DELETE	invalid request	delete identified resource

4.2. REST requests

4.2.1. Query parameters

Requests are made to the URL that identifies the correct resource, using one of the described methods. Additional parameters are passed in the query string of the request.

Parameter	Value	Remarks
format	'json' (default) or 'html'	'html' returns documentation rather than data
token	the authentication token	mandatory
fields[]	limit the returned data	may occur more than once

4.2.2. Request body

In case of POST or PUT requests, the actual resource data are passed in the request body. These data need to be sent as JSON. In case of GET and DELETE requests, the request body should not be present.

In Appendix 6.1 the data types supported by the REST API are described.

4.3. REST responses

4.3.1. Response codes

The REST API responds using the following response codes.

Code	Message	Meaning
200	Ok	The request completed succesfully
304	Not modified	The request returns the same data as a previous GET request with the same properties.
400	Bad request	The request is incorrectly formulated.
401	Unauthorized	The request is not done with sufficient authorization.
403	Forbidden	The request is not allowed access to the resource.
404	Not found	The requested resource is not present.
405	Method not allowed	The requested operation is not allowed.
410	Gone	The requested resource is not present, and no alternative location is known.
423	Locked	The requested resource is present, but access to the resource is not possible at the time of the request.
500	Internal server error	An error occurred while processing the request.
501	Not implemented	The requested method is not implemented in the API.
503	Service unavailable	The REST interface is not accessible at the time of the request.

The exact nature of any errors (codes in the 400 range) may depend on the exact request.

4.3.2. Headers

REST responses provide the usual HTTP response headers, and the REST interface may add custom headers.

Header	Content
X-Resource	The URL of the resource that was created or modified.
X-Clang-API-Error	Additional information in case an error was returned.

In case of POST and PUT requests, no data are returned in the response. If the inserted or updated data are required, an additional GET request should be performed to the URL provided in the *X-Resource* header.

4.3.3. Body

Only in the case of GET requests will a REST response contain data in the response body, in the form of JSON. The data returned by the REST interface will contain the information described in the data model, and add metadata that can be recognized by the prefix '*clang_*' to the field names. The following metadata may be present in the data:

Field	Value
clang_id	The id of the resource, which may be used in request URLs.
clang_createdat	The timestamp of creation of the resource.
clang_createdby	The Clang user that created the resource, often the user associated with the API token.
clang_modifiedat	The timestamp of the last modification of the resource.
clang_modifiedby	The Clang user that last modified the resource, often the user associated with the API token.

Note that you can't set or modify metadata, it is maintained automatically by the API.

5. Examples

This chapter describes the step-by-step process of maintaining data in the data model using examples of requests and responses. In the requests we make use of a dummy access token and a non-existing domain name. Please replace these with the values appropriate for your environment.

5.1. Data model description

The structure of the data model can be directly requested using the REST API. A GET request with a '*format*' query parameter containing '*html*' and a valid token will return a description of the data model. See appendix 6.2 for the documentat of the model used in this manual. This document was requested with the URL

```
https://.../dataextension?format=html&token=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

For brevity we will be shortening the access token in further examples.

5.2. Simple data requests

Assuming we're starting out with a completely empty data model, we need to add data before we can continue. Inserting is done by using POST requests. First we're adding some pizzas to the menu by using a POST request to the URL

```
https://.../dataextension/pizza?token=xxxxxxxx
```

The body consists of the following JSON:

```
{
      “name”: “Napolitana”
}
```

If this request is completed successfully, the response will contain an *X-Resource* header:

```
X-Resource: https://.../dataextension/pizza/clang_523c0bc14a1bb?format=json
```

Note the id in the URL. We can now retrieve the data from the model using a GET request this URL. We can omit the '*format'* header, since JSON is the default format, but we need to add our token:

```
https://.../dataextension/pizza/clang_523c0bc14a1bb?token=xxxxxxxx
```

The response contains the information we sent, and the metadata the API added:

```
{
    "name":"Napolitana",
    "clang_id":"clang_523c0bc14a1bb",
    "clang_createdat":"2013-09-20 10:48:01",
    "clang_createdby":"User",
    "clang_modifiedat":"2013-09-20 10:48:01",
    "clang_modifiedby":"User"
}
```

Suppose we wanted to change the name of this pizza. We would use the exact same URL, but in stead of a GET we would use a PUT request. In the body we pass any properties that we wish to change:

```
{
      “name”: “Quattro Formaggi”
}
```

If the request is processed correctly, we receive another response with a 200 code and the same resource URL. Another GET request will confirm that the information has actually been updated:

```
{
    "clang_createdat":"2013-09-20 10:48:01",
    "clang_createdby":"User",
    "clang_id":"clang_523c0bc14a1bb",
    "clang_modifiedat":"2013-09-20 11:19:02",
    "clang_modifiedby":"User",
    "name":"Quattro Formaggi"
}
```

Finally, if we wanted to remove the resource, we could use the URL again, but this time with a DELETE request. This request will return no data or resource headers. The success of the request is indicated by the response code. To confirm this, we can test whether the resource has been removed by doing another GET request to the resource URL. This will return a response code 404, with an additional response header

```
X-Clang-API-Error: Resource not found: {"pizza":"clang_523c0bc14a1bb"}
```

Note that we can also do a GET request without specifying the id of the resource. The response will consist of all data that the URL refers to. This may return a very large amount of data, and therefore should be used with care.

5.3. Complex data requests

The examples above are for a simple table, but can be extended to more complex situations with related

tables. As an example, let's insert an order. Assuming there is a customer in the Clang CRM with id *42* and a set of pizzas in our data model, we can compose the following request body:

```
{
    "address": "My place",
    "remarks": "Bang on the door",
    "delivered": false,
    "orderedpizza" : [
        {
            "pizza": "Napolitana",
            "number": 1,
            "remarks": "Hold the olives!"
        },
        {
            "pizza": "Quattro Stagioni",
            "number": 2
        }
    ]
}
```

In this order we set the properties of the order itself, but also immediately add two pizzas to the order. A few important observations apply to the request body:

- We can refer to records in the *lookup* table containing the pizzas themselves by the value of the field in the *lookup* table. We created a *lookup* relation between the *pizza* field in the *orderedpizza* table and the *name* field in the *pizza* table. By refering to the pizza by name in the order, the data model will create the correct reference.
- We may omit any fields that we don't need, such as the remarks field in the second ordered pizza. Note that the reference field for the *lookup* table may also be omitted. The request will succeed, and the record will be inserted, but any attempt to look up the related data will fail.

If we do a POST request to the following URL

```
https://.../dataextension/customer/clang_42/order?token=xxxxxxxx
```

the order is added to the customer record. A GET request to the returned resource URL confirms this:

```
{
    "address":"My place",
    "remarks":"Bang on the door",
    "delivered":"FALSE",
    "orderedpizza":[
        {
            "pizza":"clang_523ae32af3d75",
            "number":1,
            "remarks":"Hold the olives!",
            "clang_id":"clang_523c3543dc383"
        },
        {
            "pizza":"clang_523ae358a9142",
            "number":2,
            "clang_id":"clang_523c3543dc3ec"
```

```
        }],
    "clang_id":"clang_523c3543dc44d"
}
```

Note how the references to the pizzas have been resolved. Actually, we also could have used the ids of the pizzas while inserting the order. Now that the order has an id that is returned in the *X-Resource* header we can add an *orderedpizza* record to this order by doing a POST request to the following URL

```
https://.../dataextension/customer/clang_42/order/clang_523c3543dc44d/orderedpizza?
token=xxxxxxxx
```

with the body of the request containing just the *orderedpizza* record:

```
{
    "pizza":"Margherita",
    "number":1,
    “remarks”: “Extra anchovies”
}
```

The returned *X-Resource* will point to the inserted *orderedpizza* record. Updating or deleting the data works as demonstrated earlier. For example, updating the number for one of the ordered pizzas can be done by addressing the ordered pizza in the URL

```
https://.../dataextension/customer/clang_42/order/clang_523c3543dc44d/orderedpizza/cla
ng_523c3543dc3ec?token=xxxxxxxx
```

and updating the properties by PUT-ting the following data:

```
{
    "pizza":"Quattro Stagioni",
    "number":3,
}
```

Deleting this order can be done by using the same URL and sending a DELETE request.

5.4. Restricting selected data

In GET request the total returned set of information may become very large. Therefore it is possible to specify the fields that a GET request should be restricted to, by using one or mode '*fields[]'* parameters on the query string. For example, using the following URL

```
https://.../dataextension/customer?
token=xxxxxxxx&fields[]=order.orderedpizza.remarks&fields[]=order.orderedpizza.pizza
```

will return a list of all ordered pizzas from all customers, including remarks. Since the '*clang_id'* fields are not selected, the complete dataset is anonymized. If no '*fields[]*' parameters are present on the query string, all fields will be returned.

6. Appendices

6.1. Supported data types

Data type	Examples	Remarks
String	"This is a string."	Limited to 1048576 characters.
Date	"2013-09-23" "October 1, 2013"	See http://www.php.net/manual/en/datetime.formats.php for supported formats.
Time	"23:25:00" "4PM"	See http://www.php.net/manual/en/datetime.formats.php for supported formats.
Date and time	"2013-09-23, 16:00:00" "October 1, 2013, 4PM"	See http://www.php.net/manual/en/datetime.formats.php for supported formats.
Yes/No	1 "true"	Boolean type: 1, "true", "on", and "yes" resolve to *true*, all other values resolve to *false*.
Number	42	Signed integer number.
Decimal number	3.1415	Real number with decimal point.

Request data are validated against the defined types, and the REST API will return an error when validation fails. The *X-Clang-API-Error* will contain more information on the offending data.

6.2. Data model example documentation

When the documentation for the model described in this manual is requested from the API, the following information is returned.

Extended data model description for brand Example

Table: customer

This is a system table which has no directly accessible data.

Defined relations

- Table customer contains one or more entries from table order

Table: order

Defined columns

Name	Type	Description
address	string	
remarks	string	
delivered	boolean	

Defined relations

- Table order contains one or more entries from table orderedpizza
- Table customer contains one or more entries from table order

Table: orderedpizza

Defined columns

Name	Type	Description
pizza	string	
number	number	
remarks	string	

Defined relations

- Table order contains one or more entries from table orderedpizza
- Column pizza in table orderedpizza refers to column name in table pizza

Table: pizza

Defined columns

Name	Type	Description
name	string	

Defined relations

- Column pizza in table orderedpizza refers to column name in table pizza