Customer Profile Server REST API

Version History

Date	Export Version	Description
7/19/17	1.01	Initial Version

Introduction

REST is a style of API for interaction with remote services designed around the model that underlies the web. Interactions are based on the CRUD model, where the operations are represented via the basic HTTP POST/GET/PUT/DELETE operations. This allows APIs to use standard web infrastructure, both for development and operation. For example, since HTTP is used for transport, additional firewall openings are not generally required, and common hardware and software for HTTPS connections can be used for security.

Requirements on Requests

As noted above, requests made to a RESTful API are ordinary HTTP requests. There are however, two header requirements:

The HTTP "Authentication" header must contain a “bearer token” created via OAuth as described below,

The “Content-type” header must be “application/json”

Wrapper

The API document format has a top level wrapper which provides a uniform representation for clients to retrieve and post content, and helps to disambiguate metadata from customer data.

Data

Customer data is wrapped in a field named 'data' in the wrapper.

Example

```
HTTP/1.1 200 OK
…headers…

{
      …wrapper data…
      "data" : …
}
```

Timestamps

Timestamps are formatted using the JavaScript-compatible ISO 8601 format: yyyy-MM-dd'T'HH:mm:ss.SSSX with an offset of 0 (Z).

Errors

If a RESTful operation cannot be completed successfully, the call will return an HTTP error, and a JSON object containing the error description.

Example

```
HTTP/1.1 404 Not Found
…headers…

{
      "error": {
            "description" : "account not found"
      }
}
```

Authentication

REST API calls are protected using OAuth 2.0 tokens. For server to server calls, tokens are obtained by submitting a request to the /services/oauth/token endpoint using the client credentials grant type, and are passed as Bearer tokens.

Example token request

```
$ curl --data
"grant_type=client_credentials&client_id=6gsz65tSSUDKeM7P&client_secret
=oSOLAwb7yfNYi1nG" https://profile.example.com/services/oauth/token
{ "access_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWwuc3RhZ
2UuY3JhaW5zZGV0cm9pdC5jb20vIiwiZXhwIjoiMTQ1NzEyMzEzNiIsIm5iZiI6IjE0NTcx
MjMxMjYiLCJpYXQiOiIxNDU3MTIzMTI2Iiwic2NvcGUiOiIifQ==.AWkxj6KIt30EqKpr8j
tFcZO_jLD6VUcjbLj3iEzc5Gg=" }
```

Customer Identifiers

The profile server uses separate customer identifiers for each client to prevent clients from tracking customers across services; additionally, they may be of limited duration.

Customer Profile Information

Creating Customer Profiles

To create a customer profile, customer profile information is POSTed to the end point /services/create:

```
https://profile.example.com/services/create
```

With a JSON payload of profile values:

```
{
	"data": {
		"email": {
			"value": "sandy@clickshare.com",
			"timestamp": "2017-02-28T15:57:24.000Z"
		},
		"postalCode": {
			"value": "01002",
			"timestamp": "2017-02-28T15:57:24.000Z"
		},
		"interestHorses": {
			"value": true,
			"timestamp": "2017-02-28T15:57:24.000Z",
			"public": true
		}
		"interestGolf": {
			"value": false,
			"timestamp": "2017-02-28T15:57:24.000Z",
			"public": true
		}
	}
}
```

The request will result in either a response containing a profile home customer Id , or a response containing an error if the payload contains invalid profile values.

```
{
	"customerId": "1a2b3c",
	"data": {
	}
}
```

Updating Customer Profiles

To update a known customer profile, the customer profile information is PUT to the end point /services/update with the profile home customer Id:

https://profile.example.com/services/update/1a2b3c

With a JSON payload of profile values:

```
{
	"data": {
		"interestPolitics": {
			"value": true,
			"timestamp": "2017-02-28T15:57:24.000Z",
			"public": true
		}
	}
}
```

The request will result in either an null data response, or a response containing an error if the payload contains invalid profile values.

Customer Engagement

During the normal interaction between the customer and the web, the customer's actions provide an insight into their interests, and clients may report these interactions to the profile server for aggregation and analysis. Engagement actions are reported with a POST to the end point /services/engagement with a customer id:

https://profile.example.com/services/engagement/1a2b3c

And a JSON payload of profile values in the same form as update:

```
{
	"data": {
		"interestPolitics": {
			"value": true,
			"timestamp": "2017-02-28T15:57:24.000Z"
		}
	}
}
```

Member Access to Customer Information

Obtaining Customer Aliases

To protect customer privacy, when a member transmits a customer id to another party, the transmitting member must obtain an alias for the customer. To obtain an alias, a GET request is sent to the /services/alias end point with the member's id for the customer, and the member id for the receiving party:

https://profile.example.com/services/alias/1a2b3c/4d5e6f

The request will result in a response containing a customerId for the specified party:

```
{
	"customerId": "7g8h9i",
	"data":  {
	}
}
```

Aliases have a limited lifetime, and once expired, are no longer valid.

Lookup of Customer Information

Member sites may retrieve information from the profile service store for their own use. The information retrieved will be the intersection of the information that is marked as sharable in the service and of the information the member site is configured to receive. Configuration of this information is performed outside of the API.

To query the customer profile server, a GET request is submitted to the end point /services/lookup that identifies the customer by an identifier:

https://profile.example.com/services/lookup/1a2b3c

The request will result in a response containing the available requested data with associated timestamps:

```
{
	"data": {
		"email": {
			"value": "sandy@clickshare.com",
			"timestamp": "2017-02-28T15:57:24.000Z"
		},
		"postalCode": {
			"value": "01002",
			"timestamp": "2017-02-28T15:57:24.000Z"
		}
	}
}
```

Profile Queries

Profile Query Syntax

The syntax of profile queries is made up of name, value pairs. Matches are scored based on quality. Because a field name cannot appear more than once in a JSON object, the JSON array syntax is used if a field can match more than one value.

Evaluating Profile Queries

To evaluate a potential profile queries, you can PUT a candidate query to the end point /services/query:

https://profile.example.com/services/query

With a JSON containing payload of the query:

```
{
      "data": {
            "postalCode": [ "01002", "01003" ]
      }
}
```

The request will result in either a scoring of the number of customers matching at a particular confidence levels, or a response containing an error if the payload is invalid:

```
{
    "data": [ 5, 10, 25, 25 ]
}
```

Storing Profile Queries

To evaluate a potential profile queries, you can POST a candidate query to the end point /services/query with the profile query name:

https://profile.example.com/services/query/amherst

With a JSON payload of the query:

```
{
      "data": {
            "postalCode": [ "01002", "01003" ]
      }
}
```

The request will result in either an array scoring of the number of customers matching at particular confidence levels, or a response containing an error if the query already exists or the payload is invalid:

```
{
     "data": [ 5, 10, 25, 25 ]
}
```

Removing Stored Profile Queries

To remove a stored profile, a DELETE request is sent to the end point /services/query/ with the profile query name. The request will result in either an null data response, or a response containing an error if the profile query cannot be removed.

Applying Stored Profile Queries

To query the customer profile server, a query is PUT to the end point /services/match with an appended customer identifier:

https://profile.example.com/services/match/6w7x8y9z

With a JSON object containing an array of stored query names, and a payload of page specific data. If the array is empty or missing, all of the clients stored queries are consulted, and if the payload is included, the attributes override the customer's profile data for the purpose of the query scoring. (e.g., if a page is an article about politics, when placing ads on the page, the profile service might treat the customer as having expressed as interest in politics for that page):

```
{
      "query": [ "profile1", "profile2" ],
      "data": {
            "interestPolitics": true
      }
}
```

The request will result in a response including a scored list of matched profiles:

```
{
      "data":  {
            "profile1": 93,
            "profile2": 21
      }
}
```