Article
Sarah Schlegel · Sep 7, 2022 7m read

REST JSON webservices presentation

Hello Community!

This article gives an overview of the REST JSON webservices developed for TrakCare.

These webservices allow users to access TrakCare data from outside of the software, mainly through external apps.

They are developed in REST with ObjectScript, and they allow data access in four modes:

- **List**: gather a list of data based on a filter (for example all the episodes of a given patient);
- **Open**: get a specific row from its internal ID or a unique key (for example the episode number for a given episode);
- **Save**: insert a new entry in the right table and return its internal ID;
- **Update**: update an existing entry in a table based on its ID.

Access the webservices

The generic URL for the webservices is:

```
http://<adresse IP ou alias du serveur TrakCare>:57772/trakRestWS/tcrest/call
```

The requests are sent by POST method, with a JSON body containing the user's authentication data, the identification of the webservice to be called (class and method) and the method parameters. These parameters are, most of the time:

- for List webservices, the episode number;
- for Open, the internal ID of the element;
- for Save, the mandatory fields for inserting a new element, plus eventually the associated ACN context and action, and the non-mandatory parameters;
- for Update, the internal ID of the element and the fields to update.

The JSON structure is the following:

```
{
    "ClassName": "<Full class name>",
    "MethodName": "<Method name, generally List, Open, Save, SaveWithACN or Update>",
    "AppName": "<Name of the application calling the webservice>",
    "Request": {
        "UserName": "<TrakCare username>",
        "Password": "<Encoded password>",
```

```
        "Encoder": "Base64",
        "Params": {
            <Parameters>
        }
    }
}
```

The Encoder parameter defines the encoding of the password, which prevents sending the user's password unencrypted. But if the Encoder parameter is not defined, the Password field will contain the non-encoded string of the user's password.

Filters

The List and Open webservices often return large volumes of data, which may cause client applications to slow down. Thus, a filtering system of the webservice output is possible if an additional parameter is added to the JSON.

Filtering is possible by getting only a list of requested fields with the **FilterInclude** parameter or excluding the non-wanted fields with **FilterExclude**. The previous JSON can therefore be one of the two following:

```
{
    "ClassName": "<Full class name>",
    "MethodName": "<Method name, generally List, Open, Save, SaveWithACN or Update>",
    "AppName": "<Name of the application calling the webservice>",
    "Request": {
        "UserName": "<TrakCare username>",
        "Password": "<Encoded password>",
        "Encoder": "Base64",
        "Params": {
            <Parameters>
        },
        "FilterInclude": "<Fields to include separated by a comma>"
    }
}
```

OR

```
{
    "ClassName": "<Full class name>",
    "MethodName": "<Method name, generally List, Open, Save, SaveWithACN or Update>",
    "AppName": "<Name of the application calling the webservice>",
    "Request": {
        "UserName": "<TrakCare username>",
        "Password": "<Encoded password>",
        "Encoder": "Base64",
        "Params": {
            <Parameters>
        },
        "FilterExclude": "<Fields to exclude separated by a comma>"
    }
}
```

Code tables / Thesauri

Code tables, or thesauri, store some of the data from TrakCare. These are dictionaries referenced by other TrakCare tables by their code or description, and therefore are necessary for webservices to function properly.

Code tables list

The code table list is available for the user by using the following JSON:

```
{
    "ClassName": "Region.FRXX.WebServices.REST.CodeTables.GenericCodeTables",
    "MethodName": "List",
    "AppName": "<Name of the app calling the webservice>",
    "Request": {
        "UserName": "<TrakCare username>",
        "Password": "<Password>",
        "Params": {
            "CT": "*"
        }
    }
}
```

And the output JSON will contain the list or the available thesauri.

Code table content

To access a given code table, the CT parameter in the previous JSON has to contain the name of the code table we wish to access. Two additional parameters, **Filter and FilterActif**, specify the request.

Filter

The **Filter** parameter filters on the code and description of the elements in the code table. It is based on a pattern matching system similar to SQL's, using the % character as a wildcard.

The filtering can be done in one of the following ways:

- Strict filtering with an exact string: "Bilateral" will only output the elements whose code or description matches "Bilateral", non-case sensitive.
- Soft filtering with the % wildcard: the % defines an unknown group of characters. Thus,
    - "%al" will output all the elements finishing by "al" (or "AL" or "Al" or "aL")
    - "Bi%" will output all the elements starting with "Bi" (or "BI" or "bi" or "bI").
    - "%la%" will output all elements whose code or description contains "al" somewhere (non-case sensitive).
        - in this case, the % can be combined to match elements containing multiple strings like for example "%bi%al%"
    - "%" will output all the elements. But warning, some code tables contain a lot of rows, which may cause the client application to crash or slow down.

If the Filter parameter is not defined or empty, the webservice outputs an empty result.

FilterActif

By default, the webservice only outputs the default parameters of the thesaurus. To filter on active and inactive elements, the parameter **FilterActif** set to the value "ALL" has to be passed in the JSON.

A complete example to retrieve the care providers list (CTCareProv code table) is:

```
{
    "ClassName": "Region.FRXX.WebServices.REST.CodeTables.GenericCodeTables",
    "MethodName": "List",
    "AppName": "Postman",
    "Request": {
        "UserName": "userWS",
        "Password": "****",
        "Params": {
            "CT": "CTCareProv",
            "Filter": "%",
            "FilterActif": "ALL"
        }
    }
}
```

ACN

The Save method is a generic method implemented in all webservices which inserts a new entry. Nevertheless, a Save only inserts the entry in the TrakCare database and not in the EPR.

The **SaveWithACN** method implemented by all Save webservices saves the data in the database and the EPR. This method requires two additional parameters: **ACNContext** and **ACNAction**. The episode number (**Episode**) also has to be defined, even if it isn't required for normal inserts.

All the contexts can be retrieved in the code table **MRCEncEntryType**. Then, the associated action can be retrieved by calling the generic Save webservice and its **GetAction** method.

Final examples

Let's consider the family medical history. A request to list all the family's medical history of a patient will be as follows:

```
{
    "ClassName": "Region.FRXX.WebServices.REST.PAFamily.List",
    "MethodName": "List",
    "AppName": "SoapUI",
    "Request": {
        "UserName": "userWS",
        "Password": "******",
        "Params": {
            "IPP": "21002205"
        }
    }
}
```

To then retrieve one of these family medical records, we will use:

```
{
    "ClassName": "Region.FRXX.WebServices.REST.PAFamily.Open",
    "MethodName": "Open",
    "AppName": "SoapUI",
    "Request": {
        "UserName": "userWS",
        "Password": "demo",
        "Params": {
            "ID": "679063||2"
        }
    }
}
```

To save a new medical history record in the EPR, we will have to use the following JSON:

```
{
    "ClassName": "Region.FRXX.WebServices.REST.PAFamily.Save",
    "MethodName": "SaveWithACN",
    "AppName": "SoapUI",
    "Request": {
        "UserName": "userWS",
        "Password": "demo",
        "Params": {
            "Episode": "196001820",
            "IPP": "21002205",
            "ACNContext": "Toutes les actions",
            "ACNAction": "TC.FMHIS",
            "MRCIDCodeOrDesc": "C254",
            "FAMRelationCodeOrDesc": "10",
            "FAMDesc": "Created with a webservice"
        }
    }
}
```

And finally, to update that same record, we will only send the modified fields to the Update method:

```
{
    "ClassName": "Region.FRXX.WebServices.REST.PAFamily.Save",
    "MethodName": "Update",
    "AppName": "SoapUI",
    "Request": {
        "UserName": "userWS",
        "Password": "******",
        "Params": {
            "ID": "679063||1",
            "FAMDesc": "Updated with a webservice"
        }
    }
}
```

Conclusion

These REST JSON webservices allow third-party applications to access and modify TrakCare data.

The next step would be to add a layer on the redirection Engine to let users access the data with the REST standards (resource separation by URL, use of HTTP verbs, etc.)

#JSON #ObjectScript #REST API #TrakCare