Rockland Technical Note 057

— Rockland API—
Customer Facing Document

JP Juteau

2022-10-26

Rockland Scientific International Inc.
520 Dupplin Rd
Victoria, BC, CANADA, V8Z 1C1
www.rocklandscientific.com

Copyright Notice

info@rocklandscientific.com
tel: +1 250 370 1688

Version History

Date	Description
2022-05-20	Initial creation
2022-10-12	Updating information to new API framework
2022-10-26	Adding Project/Edit information

Contents

List of Tables

ROCKLAND SCIENTIFIC

1 Introduction

This document provides a short guide to interacting with the Rockland API for the purpose of decoding Rockland q-files and extracting your decoded data from our database. To do so, the document will describe:

1. Log in and authentication
2. Uploading Slocum **.mri* files
3. Downloading data types.

The Rockland API is accessible at the following URL: `https://rocklandapi.azurewebsites.net`

In order to connect to the Rockland API, you will need to create an account in the Rockland Online Portal. An email should have been sent (perhaps received in your junk folder) with a link to login and create an account with its credentials. Those same credentials (email address and password) will be used to connect to the Rockland API.

If you experience any issues setting up your user account in the Rockland Online Portal, please contact zissou@rocklandscientific.com.

To facilitate the integration, the text below provides snippets of Python code to perform the API queries. This code was auto-generated and should be tested before being used.

1.1 Workflow

The expected workflow for interacting with the API is described below:

1. Confirm your credentials are working
2. Create a Project to store your data
3. Data files are uploaded to the Project as ”Profiles”
4. The decoded data from Profiles can be queried from the API

Project Creation

A project must be created to hold your microstructure data. The Project requires a name and an instrument serial number to be created. If the project is successfully created, a token will be returned to uniquely identify the project for further requests.

Uploading data

Once a project has been created, your q-files can be uploaded to the project. This requires uploading the `*.mri` or `*.q` file to create a ”Profile”.

During the process, the data will be parsed from the file and stored in the database. The data will be accessible using the profile identification token returned by the endpoint.

Retrieving data

Finally, to extract data from the database, the easiest method is to query the data using profiles and data identifiers. Each profile has a unique token and each data types (*e.g.* ϵ_1, $\Phi_{\nabla u1}(f)$, etc.) has a unique hexadecimal identifier.

2 Logging into the Rockland API

To perform most requests in the Rockland API, an authentication token is required which identifies you as a user. The first step to interact and query the API is to log-in *i.e.* get an authentication token. To do so, you will need:

- Your Rockland Online Portal username (email address)
- Your Rockland Online Portal password (created when first logging into the Rockland Online Portal)
- A Source Token (`c6X2ADq9YqA3Hh5`)
- Your organization name as presented in the Rockland Online Portal (Oregon State University)

The code below provides an example of such a query using Python's `http.client` package.

```
import http.client
import json

conn = http.client.HTTPSConnection("rocklandapi.azurewebsites.net")

payload = json.dumps({
        "username": "your_username@mail.com",
        "password": "your_password",
        "sourceToken": "c6X2ADq9YqA3Hh5",
        "OrganizationName": "Oregon State University"
})
headers = {
        'Content-Type': 'application/json'
}
conn.request("POST", "/api/Auth/Login", payload, headers)
result = conn.getresponse()
```

The expected response includes an authentication token and a tokenExpiry. The token includes information about your user identity and your organization. For security purposes, the token is set to expire an hour after the log in (note that times are set in UTC).


```
"message": "",
"body": {
    "token": "eyJhbGciOiJIUzI1BgLsPnT5aCI5IstDGFK8.eyJzbX8GW86iJaaXNzb3VfQWlyZHJvcCIsInB0I
joiMmQ5ODE2YmY0OTkxNDUxOWEyMzE3M2E0ZjlmYmRhODciLCJleHAiOjE2NTk3MTU3MTMsImlzcyI6Imh0dHA
6Ly9yb2NrbGFuZGFwaS5henVyZXdlYnNpB07EQ0G2B1XVTWxnaweb7Ctc8InF1ZCI6ImFhNTQyZmZjLWU4NjIt
NDEwNy05NzY4YmIwN2M5YiJ9bFwE7QATquWW14ID7MA35iPhp2a5sSfozeLfMHyR4-I",
    "tokenExpiry": "2022-08-05T16:08:33.2058928Z",
    "message": ""
}
```


ROCKLAND SCIENTIFIC

3 Create Project

The Project/Create endpoint will create a new empty project. To create the project you will need to provide the following:

- Your Rockland authentication token, outputted by the Login endpoint
- A name for the project
- An optional description for the project
- The serial number of the instrument used to collect data for the project

The code below provides an example of such a query using Python's `http.client` package. The payload and token variables need to be edited.

```
import http.client
import json

conn = http.client.HTTPSConnection("rocklandapi.azurewebsites.net")

payload = json.dumps({
    "Name": "Project_Name",
    "Description": "Project_Description",
    "DataType": "RDL_isdp",
    "Instruments": [
        {
            "InstrumentSN": "SN123"
        }
    ]
})
token = "eyJhbGciOiJIUzI1BgLsPnT5aCI5IstDGFK8.eyJzbX8GW86iJaaX
Nzb3VfQWlyZHJvcCIsInB0IjoiMmQ5ODE2YmY0OTkxNDUxOWEyMzE3M2E0ZjlmYmRh
ODciLCJleHAiOjE2NTk3MTU3MTMsImlzcyI6Imh0dHA6Ly9yb2NrbGFuZGFwaS5hen
VyZXdlYnNpB07EQ0G2B1XVTWxnaweb7Ctc8InF1ZCI6ImFhNTQyZmZjLWU4NjItNDE
wNy05NzY4YmIwN2M5YiJ9bFwE7QATquWW14ID7MA35iPhp2a5sSfozeLfMHyR4-I"

headers = {
    'Authorization': 'Bearer ' + token,
    'Content-Type': 'application/json'
}
conn.request("POST", "/api/Project/Create", payload, headers)
result = conn.getresponse()
```

The expected response includes a project token that can be used to add profiles to the project.

```
"message": "",
"body": "f8b45014-1606-4fbe-94f7-bdd9b24c802e"
```

4 Edit Project

Once a project has been created, it can be edited to correct some of its information. The input structure for the edit endpoint is identical to the project creation, however, it is only necessary to provide the parameters you wish to modify. To edit a project, you will require:

- Your Rockland authentication token (JWT) outputted by the Login endpoint
- A Project token for the Project you wish to edit

The code below provides an example of such a query using Python's `http.client` package.

```
import http.client
import json

conn = http.client.HTTPSConnection("rocklandapi.azurewebsites.net")
payload = json.dumps({
        "name": "New name",
        "description": "New project description",
        "token": "18c4adc4-5bc1-46d9-bbc1-35483d002092"
})
token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzb3VyY2UiOiJBZD
wdUdGMEJLcThaV0oiLCJwdCI6IjJkOTgxNmJmNDk5MTQ1MTlhMjMxNzNhNGY5ZmJ
kYTg3IiwiZXhwIjoxNjY2ODE4MDU1LCJpc3MiOiJodHRwOi8vcm9ja2xhbmRhcGk
uYXp1cmV3ZWJzaXRlcy5uZXQiLCJhdWQiOiJhYTU0MmZmYy1lODYyLTQxMDctOTc
wYi0xNDZmOGJiMDdjOWIifQ.WAEcqSgDbg5UJeaYy180qe78D9BIFtNG5SWH2CTg"

headers = {
        'Authorization': 'Bearer ' + token,
        'Content-Type': 'application/json',
}
conn.request("POST", "/api/Project/Edit", payload, headers)
res = conn.getresponse()
```


ROCKLAND SCIENTIFIC

5 Upload data

Uploading data files to the Rockland database consists in creating new profiles in the project. This is doone using the Profile/New endpoint.

To call the endpoint you will require the following:

- Your Rockland authentication token (JWT) outputted by the Login endpoint
- A Project token from the Project/Create endpoint
- A data file (.mri or *.q)

The code below provides an example of such a query using Python's `http.client` package. The filePath, projectToken and token variables need to be edited.

```
import http.client
from codecs import encode

filePath = 'path/to/file.mri'

projectToken = "f8b45014-1606-4fbe-94f7-bdd9b24c802e"

token = "eyJhbGciOiJIUzI1BgLsPnT5aCI5IstDGFK8.eyJzbX8GW86iJaaX
Nzb3VfQWlyZHJvcCIsInB0IjoiMmQ5ODE2YmY0OTkxNDUxOWEyMzE3M2E0ZjlmYmRh
ODciLCJleHAiOjE2NTk3MTU3MTMsImlzcyI6Imh0dHA6Ly9yb2NrbGFuZGFwaS5hen
VyZXdlYnNpB07EQ0G2B1XVTWxnaweb7Ctc8InF1ZCI6ImFhNTQyZmZjLWU4NjItNDE
wNy05NzY4YmIwN2M5YiJ9bFwE7QATquWW14ID7MA35iPhp2a5sSfozeLfMHyR4-I"


fileType = "RDL_isdp"

conn = http.client.HTTPSConnection("rocklandapi.azurewebsites.net")

dataList = []
boundary = 'wL36Yn8afVp8Ag7AmP8qZ0SA4n1v9T'
dataList.append(encode('--' + boundary))

dataList.append(encode(
    'Content-Disposition: form-data; name=; filename=file'))
dataList.append(encode('Content-Type: application/octet-stream'))
dataList.append(encode(''))
with open(filePath, 'rb') as f:
dataList.append(f.read())

dataList.append(encode('--' + boundary))

dataList.append(encode(
```


ROCKLAND SCIENTIFIC

```
    'Content-Disposition: form-data; name=ProjectToken;'))
dataList.append(encode('Content-Type: text/plain'))
dataList.append(encode(''))
dataList.append(encode(projectToken))

dataList.append(encode('--' + boundary))

dataList.append(encode(
    'Content-Disposition: form-data; name=FileType;'))
dataList.append(encode('Content-Type: text/plain'))
dataList.append(encode(''))
dataList.append(encode(fileType))

dataList.append(encode('--'+boundary+'--'))
dataList.append(encode(''))
body = b'\r\n'.join(dataList)
payload = body
headers = {
        'Authorization': 'Bearer ' + token,
        'Content-type': 'multipart/form-data; boundary={}'.format(boundary)
}
conn.request("POST", "/api/Profile/New", payload, headers)
result = conn.getresponse()
```

If the profile creation was successful, the endpoint will return a success message as follows:

```
"message": "Success",
"body": null
```

6 Get Data

To get your data from one or multiple profiles, you can call the Data/Get endpoint.

To call the endpoints you will require the following:

- Your Rockland authentication token (JWT) outputted by the Login endpoint
- A Project token from the Project/Create endpoint
- The profileToken(s) from the profile you wish to query
- The data type Id(s) you wish to query.

The code below provides an example of such a query using Python's `http.client` package. Note that the projectToken is required by the profileTokens and dataTypes can be empty. In such a case, all the profiles and / or all the data types will be returned by the query. Both the profileTokens and data type Ids are expected to coma delimited strings with the desired values.

A list of the different data types Ids and their corresponding channels is available in Appendix A

```
import http.client

token = "eyJhbGciOiJIUzI1BgLsPnT5aCI5IstDGFK8.eyJzbX8GW86iJaaX
Nzb3VfQWlyZHJvcCIsInB0IjoiMmQ5ODE2YmY0OTkxNDUxOWEyMzE3M2E0ZjlmYmRh
ODciLCJleHAiOjE2NTk3MTU3MTMsImlzcyI6Imh0dHA6Ly9yb2NrbGFuZGFwaS5hen
VyZXdlYnNpB07EQ0G2B1XVTWxnaweb7Ctc8InF1ZCI6ImFhNTQyZmZjLWU4NjItNDE
wNy05NzY4YmIwN2M5YiJ9bFwE7QATquWW14ID7MA35iPhp2a5sSfozeLfMHyR4-I"

projectToken  = "f8b45014-1606-4fbe-94f7-bdd9b24c802e"
profileTokens = "e96ccfc0,3e251bc3";
dataTypeIds   = "0xA11,0xA12,0x841,0x842"


conn = http.client.HTTPSConnection("localhost", 5001)
payload = ''
headers = {
        'Authorization': 'Bearer '+token,
        'Cookie': 'allow-cookies=true'
}
conn.request("GET", "/api/Data/Get?projectToken="+ projectToken
    +"&profileTokens="+profileTokens
    +"&DataTpeIds="+ dataTypeIds, payload, headers)


res = conn.getresponse()
```

If successful the response body will contains an array of structures, one structure per profile in the project.

Each structure will contain the token of the profile, a data array for each subtype of the given data type, a time array, pressure array, and an array of data type ids corresponding to the sub-types of the data arrays:

```
"message": "",
"body": [
    {
        # First profile in the project
        [
        "profileToken": "e96ccfc0",
        "data": [
            [
                # data array of type 0xA11 (2577)

                3.394e-10,
                ...,
                2.983e-9
            ],
            [
                # data array of type 0xA12 (2578)

                4.584e-11,
                ...,
                3.689e-10
            ],
            [
            # data array of type 0x841 (2113)

            1.12,
            ...,
            2.56
            ],
            [
            # data array of type 0x842 (2114)

            0.56,
            ...,
            1.56
        ]],
        "profileToken": "3e251bc3",
        "data": [# same structure as above]
        ]
        "typeIds": [
            2577,
            2578,
```


ROCKLAND SCIENTIFIC

```
            2113,
            2114
        ]
    }
]
```

The API endpoint has the capability to return spectral data as well. In those instances, the data array will be three (3) dimensional including a dimension for the spectra.

A Data Types

The following table shows the different data types and their corresponding codes:

Kinematics Sensor and Data:

Name	Code
Time	0x100
Accelerometer	0x110 (Ax +1, Ay +2, Az +3)
Piezo	0x120 (Ax +1, Ay +2)
Inclinometer	0x130 (X +1, Y +2, T +3)
Theta	0x140
Magnetometer	0x150 (Mx +1, My +2, Mz +3)
Pressure	0x160 (P +1)

Table 1: List of data types and their corresponding ids for kinematics sensor and data

Instrument Meta Data:

Name	Code
Battery voltage	0x210
Pressure Voltage	0x220
EM Current	0x230
Position	0x240 (Lat +1, long +2, Z +3)

Table 2: List of data types and their corresponding ids for instrument meta data

Instrument Speed:

Name	Code
EM	0x310
Vector	0x320
FallRate	0x330
Glide	0x340
HotelSpeed	0x350
Speed	0x360

Table 3: List of data types and their corresponding ids for instrument speed

CT Sensor Data:

Name	Code
Temperature	0x410 (Jac +1, SeaBird +2, RBR +3)
Conductivity	0x420 (Jac +1, SeaBird +2, RBR +3)
Salinity	0x430
Density	0x440
Viscosity	0x450

Table 4: List of data types and their corresponding ids for CT sensor data

Chlorophyll, turbudity and dissolved oxygen sensors:

Name	Code
Chlorophyll	0x510
Turbidity	0x520
Dissolved Oxygen	0x530

Table 5: List of data types and their corresponding ids for chlorophyll, turbudity and dissolved oxygen sensors

Microstructure sensors:

Name	Code
Shear	0x610 (sh1...sh15 $\rightarrow$ +1...0xF)
Therm	0x620 (T1...T15 $\rightarrow$ +1...0xF)
Ucond	0x630 (C1...C15 $\rightarrow$ +1...0xF)
Grad T	0x640 (∇T1...∇T15 $\rightarrow$ +1...0xF)
Grad C	0x650 (∇C1...∇C15 $\rightarrow$ +1...0xF)

Table 6: List of data types and their corresponding ids for microstructure sensors

Shear Quality Control:

Name	Code
Kmax	0x810 (kmax_u1...kmax_u15 $\rightarrow$ +1...0xF)
Variance Resolved	0x820 (var_u1...var_u15 $\rightarrow$ +1...0xF)
MAD	0x830 ($\mathcal{E}$1 MAD...$\mathcal{E}$15 MAD $\rightarrow$ +1...0xF)
Figure of Merit	0x840 ($\mathcal{E}$1 FoM...$\mathcal{E}$15 FoM $\rightarrow$ +1...0xF)

Table 7: List of data types and their corresponding ids for shear quality control

Spectral Data:

Name	Code
Frequency	0x910
Shear Spectra	0x920 (sh1 ... sh15 $\rightarrow$ +1...0x0F)
Cleaned Shear Spectra	0x930 (sh1 ... sh15 $\rightarrow$ +1...0x0F)
Temperature Spectra	0x940 (T1 ... T15 $\rightarrow$ +1...0x0F)
Cleaned Temperature Spectra	0x950 (T1 ... T15 $\rightarrow$ +1...0x0F)
Conductivity Spectra	0x960 (C1 ... C15 $\rightarrow$ +1...0x0F)
Cleaned Conductivity Spectra	0x970 (C1 ... C15 $\rightarrow$ +1...0x0F)
Vibration Spectra	0x980

Table 8: List of data types and their corresponding ids for spectral data

Computed Values:

Name	Code
Dissipation	0xA10 (ε1...ε15 $\rightarrow$ +1...0xF)
Buoyancy Frequency	0xA20
Eddy diffusivity	0xA30

Table 9: List of data types and their corresponding ids for computed values

End of document