




Building a custom connector for Power BI that supports OAuth2 to visualize my wellness data

February 27, 2019 / Building something /  Jussi Roine



In my previous [blog.post](#), I wrote about the Oura Ring and how it tracks my wellness and activities. As part of the service Oura provides there's also a comprehensive API that can be leveraged for your own purposes.

As the data that is being tracked is very personal ("hmm, I wonder why Jussi is being active at 3:23 am.. oh he's feeding the baby"), I see little justifications (yet) for sharing this data to anyone else.

I wanted to visualize the data from Oura's Cloud using their API. But I wanted to do this in my own terms by using Microsoft's Power BI. If you're not familiar with Power BI, I suggest you take a quick look at this nice introductory video from Microsoft. It's only 90 seconds.



What is Power BI?

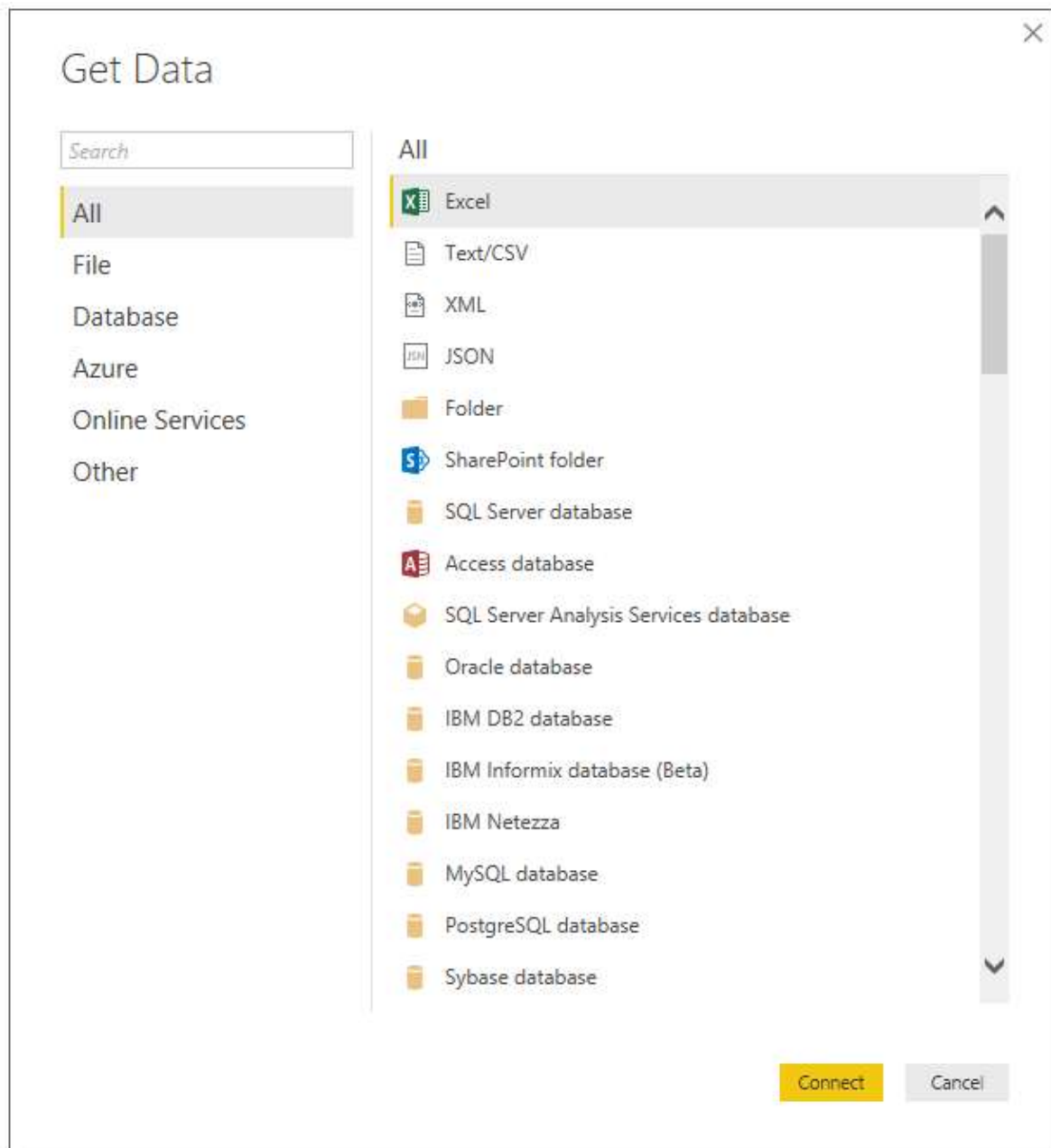
I claim not to be a Power BI expert. I'm just really, really good at poking things at random and bending them at my will in a very, very short time.

To get started, I needed [Power BI Desktop](#). This is the rich power user tool for building Power BI reports. Said reports will then pull data from systems of your choices – or APIs of your choice, and for me this would be the [Oura Ring Cloud API](#). You don't need the paid Power BI subscription, but you do need the Oura Ring to access their APIs. Makes sense, as without a ring you wouldn't have any data to work with.

Note: *The guidance below on building a custom connector for Power BI applies to any common API, so even if you don't have the Oura ring, the very same steps in this post still apply.*

Getting started: Connecting to Oura Cloud API with Power BI Desktop

Power BI Desktop has a wealth of built-in connectors. Below you'll see a sample of these. As the Oura API is producing JSON, I hoped I could simply use the JSON Connector. This unfortunately requires the .json files to be already somewhere on the disk, and I need data that will update automatically.



Power BI Desktop default connectors

I briefly considered building a simple Azure Function to wrangle the .json files but I figured there has to be a more approachable way. Turns out I need to build a Custom Connector.

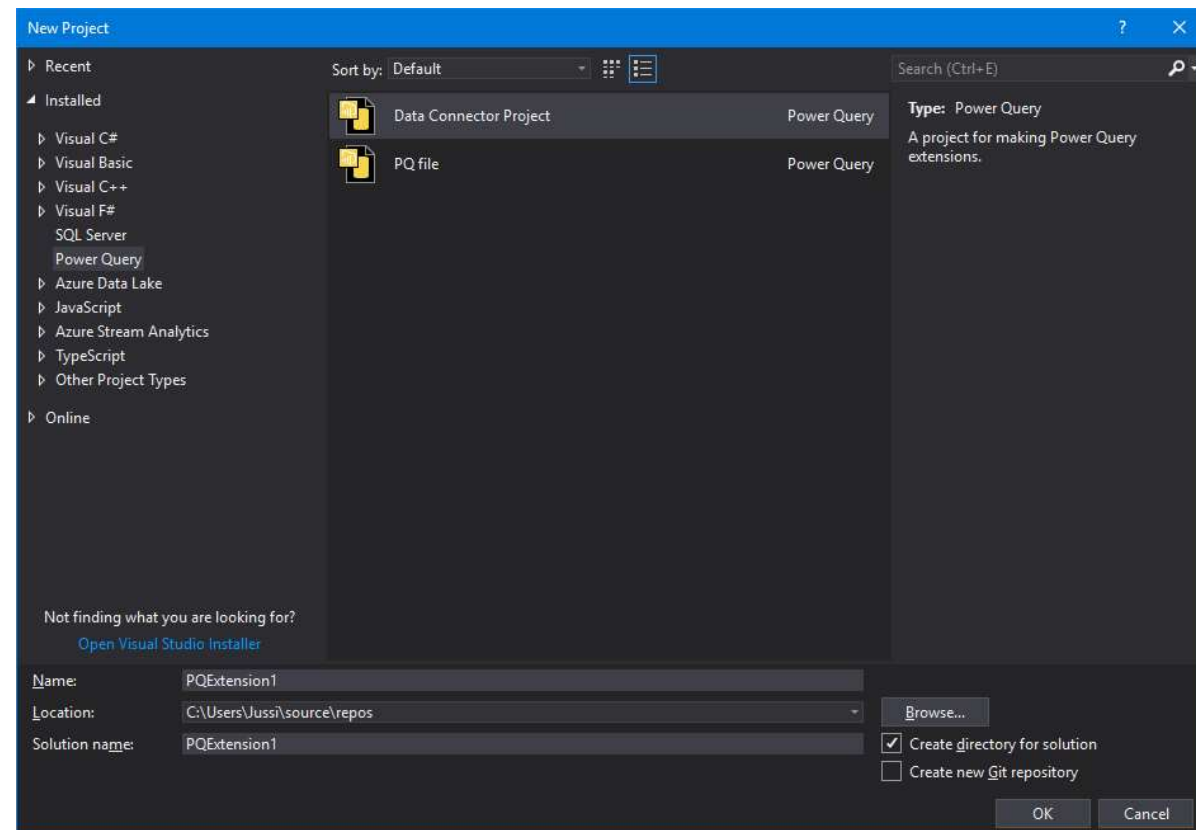
Implementing the Custom Connector for Power BI

A custom connector is something I hadn't done before. I'd seen the list of connectors grow over time, but I figured it's just Microsoft being busy and adding new connectors based on UserVoice feedback.

I found this instructional [article](#) on docs.microsoft.com to help me get started. What I needed to build was a custom connector that in turns connects with Oura's API, performs OAuth2 authorization and allows me to use the access token that is generated to perform queries. "How *hard* can it be", I remember asking myself.

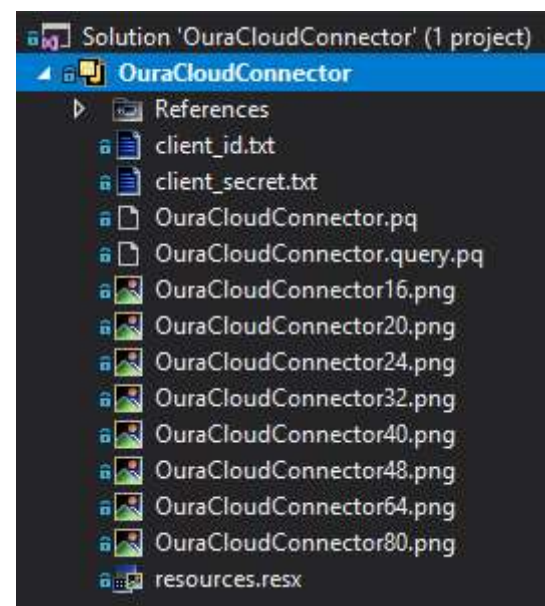
For starters, I needed Visual Studio, which I already had installed – version 2017. To develop Power BI-related projects I needed to install [Power Query SDK](#) from VS Marketplace. It's a next-next-done install.

This adds a new Project Template in Visual Studio called **Data Connector Project**. The other template, **PQ File**, is just the connector class without any fluff (but this isn't needed as the Data Connector Project already includes this too).



I'm using the dark theme to make this look more professional

For once, the project template is super simple! I've had it with all the scaffolding, generators and other weird new-school tools that produce non-intelligible files in a hundred directories. This is simple and we like simple. Simple rarely fails majestically.



Note: I've added `client_id.txt` and `client_secret.txt` manually, more on those below.

The core of the connector is in the .pq file, which I've named `OuraCloudConnector.pq`. First time opening this file I almost rolled my eyes out from my head. "What.. the..?" It looks like this:

```

[DataSource.Kind="HelloWorld", Publish="HelloWorld.Publish"]
shared HelloWorld.Contents = (optional message as text) =>
    let
        message = if (message <> null) then message else "Hello world"
    in
        message;

HelloWorld = [
    Authentication = [
        Implicit = []
    ],
    Label = Extension.LoadString("DataSourceLabel")
];

HelloWorld.Publish = [
    Beta = true,
    ButtonText = { Extension.LoadString("FormulaTitle"),
Extension.LoadString("FormulaHelp") },
    SourceImage = HelloWorld.Icons,
    SourceTypeImage = HelloWorld.Icons
];

HelloWorld.Icons = [
    Icon16 = { Extension.Contents("HelloWorld16.png"),
Extension.Contents("HelloWorld20.png"), Extension.Contents("HelloWorld24.png"),
Extension.Contents("HelloWorld32.png") },
    Icon32 = { Extension.Contents("HelloWorld32.png"),
Extension.Contents("HelloWorld40.png"), Extension.Contents("HelloWorld48.png"),
Extension.Contents("HelloWorld64.png") }
];

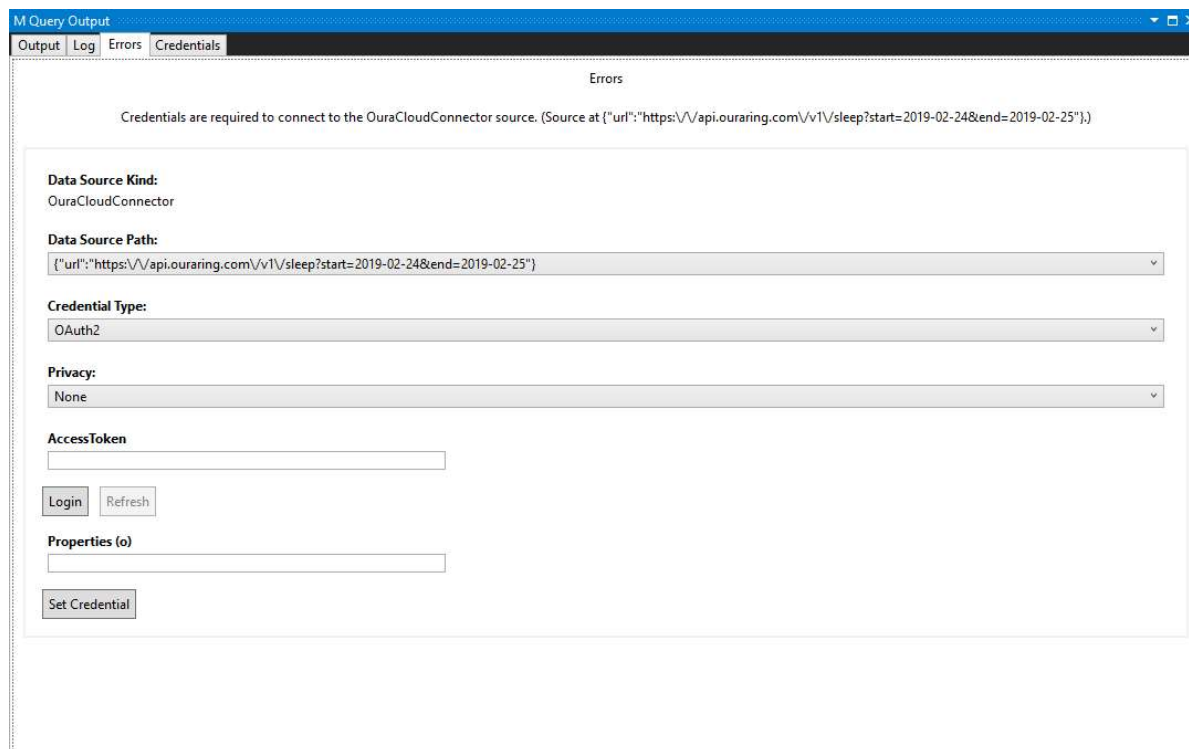
```

I had very little to start with. Upon further inspection this is [Power Query M Formula Language](#), the lingua franca of Power BI and all things Data. I think. I have zero intention in learning M any more than I need to learn Swedish [grammar](#). Based in my experience I often encounter a situation like this — I just need to learn *enough* to survive but not too much as I don't have all day.

Dissecting the sample Hello World code took about an hour and involved some creative googling. There are 4 parts to a connector:

1. Data Source definition (**DataSource.Kind**) — what does the connector *do*?
2. Authentication (**HelloWorld = [Authentication..]**) — how do we authenticate to the data source?
3. Connector details (**DataSource.Publish**)— how does the connector look like in Power BI Desktop
4. Metadata (**Icons, Window.width**)— icons, window sizing and other details

In the end, it wasn't *that* incomprehensible. It took me embarrassingly long to figure out I could just *F5* through my connector without deploying it for real. This builds the solution and runs a testing tool called M Query Output that I found very useful.



The screenshot shows the 'M Query Output' window with the 'Credentials' tab selected. The window title is 'M Query Output' and it has tabs for 'Output', 'Log', 'Errors', and 'Credentials'. The 'Errors' tab is active, displaying a message: 'Credentials are required to connect to the OuraCloudConnector source. (Source at {"url":"https://api.ouraring.com/v1/sleep?start=2019-02-24&end=2019-02-25"})'. Below the message, there are several fields and buttons:

- Data Source Kind:** OuraCloudConnector
- Data Source Path:** {"url":"https://api.ouraring.com/v1/sleep?start=2019-02-24&end=2019-02-25"}
- Credential Type:** OAuth2
- Privacy:** None
- AccessToken:** (empty text box)
- Login** and **Refresh** buttons
- Properties (o):** (empty text box)
- Set Credential** button

If you see an error it means you're about to learn something new

Based on my data source definition and authentication details, I could execute tests against a live API and run sample queries. These queries are picked from the *ProjectName.query.pq* -file:

```
let
    result = OuraCloudConnector.Contents()
in
    result
```

OuraCloudConnector.Contents() is the connector definition (Kind) and the method allows for passing parameters. Such as an URL. To my API. Makes sense, right?

Challenges ahead

I trawled through the [sample connectors](#) from GitHub that Microsoft has produced. The Microsoft Graph sample called [MyGraph](#) I found to be especially useful. Using the Hello World sample, the MyGraph sample

and a *lot* of traffic inspection with [Fiddler](#) I was able to produce a working OAuth2 implementation.

By first verifying what Oura's API is expecting I was then able to reproduce those calls manually with a browser, and then re-tracing those steps back to my M -powered custom connector. This took one evening to get to a state that I was able to get data in Power BI. I wouldn't proclaim myself as an expert on either Power BI or custom connectors but at least I have something to show in the next quarterly secret meeting of Power BI gurus (I think they have those type of meetings?).

Oura's API is pleasant to work with as it is very close to a standard OAuth2 implementation. They even have a tiny helper tool that generates clickable test links. I also found the API to be reliable – it didn't fail me once during my tests.

AUTHORIZATION URLS

These are valid Authorization urls for the application, so you can easily try them out:

SERVER-SIZE FLOW AUTHORIZATION URL

```
https://cloud.ouraring.com/oauth/authorize?state=XXX&client_id=[REDACTED]&response_type=code
```

CLIENT-SIZE FLOW AUTHORIZATION URL

```
https://cloud.ouraring.com/oauth/authorize?state=XXX&client_id=[REDACTED]&response_type=token
```

When things go wrong, you'll get raw HTTP status codes back. Helpful in a sense but upon seeing the 16th Unauthorized 401 you realize it's time to open a bottle of red.

I went back and forth for a while between successfully calling the APIs from my browser, tracing the GETs and POSTs with Fiddler and trying to replicate those calls identically from M within my custom connector and the M Query Tool. As I'm more accustomed to C#, PowerShell, plain old BAT files and other scripting interfaces it took some extra effort to bend M to my will. I probably didn't succeed in the most fashionable sense, but *it works on my machine!*

Finalizing the Custom Connector with a working OAuth2 authorization flow

I'm a little bit of proud that I got this working. Let's walk through the code — it's also available in my [Github repo](#).

First, I need a few variables to hold URIs for my OAuth2 calls:

```
client_id = Text.FromBinary(Extension.Contents("client_id.txt"));
client_secret = Text.FromBinary(Extension.Contents("client_secret.txt"));
redirect_uri = "https://oauth.powerbi.com/views/oauthredirect.html";
token_uri = "https://api.ouraring.com/oauth/token";
```

```
authorize_uri = "https://cloud.ouraring.com/oauth/authorize";  
logout_uri = "https://login.microsoftonline.com/logout.srf";
```

I need to pick OAuth2 Client ID and Client Secret (that Oura API generates for me to use) in a text file. This was the model for storing secrets all samples from Microsoft used, and I couldn't find a working and a more reasonable alternative. Keep in mind all data that is being accessed is *personal*, so I see this as an acceptable approach until a more secure approach is offered for M. These two text files simply contain the strings for Client ID and Client Secret and they do not change.

authorize_uri is used to initiate the authorization polka with the API. *token_uri* is used to retrieve an access token and *redirect_uri* is used to redirect back to Power BI Desktop. *logout_uri* I did not need but felt it would look nicer here for completeness.

Next, Oura API provides scopes but I only need the *daily* scope:

```
scope_prefix = "";  
scopes = {  
    "daily"  
};
```

And now, it's time to define the actual custom connector. What happens below is that I'm simply getting a Power Query statement and passing it as-is directly to the web and picking up – what I presume and trust – is JSON data back. It's very simple to implement in M:

```
[DataSource.Kind="OuraCloudConnector", Publish="OuraCloudConnector.Publish"]  
shared OuraCloudConnector.Contents = (url as text) =>  
    let  
        source = Json.Document(Web.Contents(url))  
    in  
        source;
```

Next, I need to implement OAuth2 tango:

```
OuraCloudConnector= [  
    TestConnection = (dataSourcePath) => { "OuraCloudConnector.Contents", dataSourcePath },  
    Authentication = [  
        OAuth = [  

```



```

        StartLogin=StartLogin,
        FinishLogin=FinishLogin,
        Refresh=Refresh,
        Logout=Logout
    ]
],
Label = Extension.LoadString("DataSourceLabel")
];

```

Authentication is set to OAuth, and I needed a few helper functions (*StartLogin*, *FinishLogin*, *Refresh* and *Logout*) to complete these. I lifted the helper functions from the [MyGraph](#) sample.

I then need to adjust the helper functions to pass only the values Oura API is expecting. For this to work, I need a few small changes in *TokenMethod* – this is called when *StartLogin()* is completed and a redirect occurs.

```

TokenMethod = (grantType, tokenField, code) =>
    let
        queryString = [
            grant_type = "authorization_code",
            redirect_uri = redirect_uri,
            client_id = client_id,
            client_secret = client_secret
        ],
        queryWithCode = Record.AddField(queryString, tokenField, code),

        tokenResponse = Web.Contents(token_uri, [
            Content = Text.ToBinary(Uri.BuildQueryString(queryWithCode)),
            Headers = [
                #"Content-type" = "application/x-www-form-urlencoded",
                #"Accept" = "application/json"
            ],
            ManualStatusHandling = {400}
        ]),
        body = Json.Document(tokenResponse),
        result = if (Record.HasFields(body, {"error", "error_description"})) then
            error Error.Record(body[error], body[error_description], body)
        else

```

```
        body
    in
        result;
```

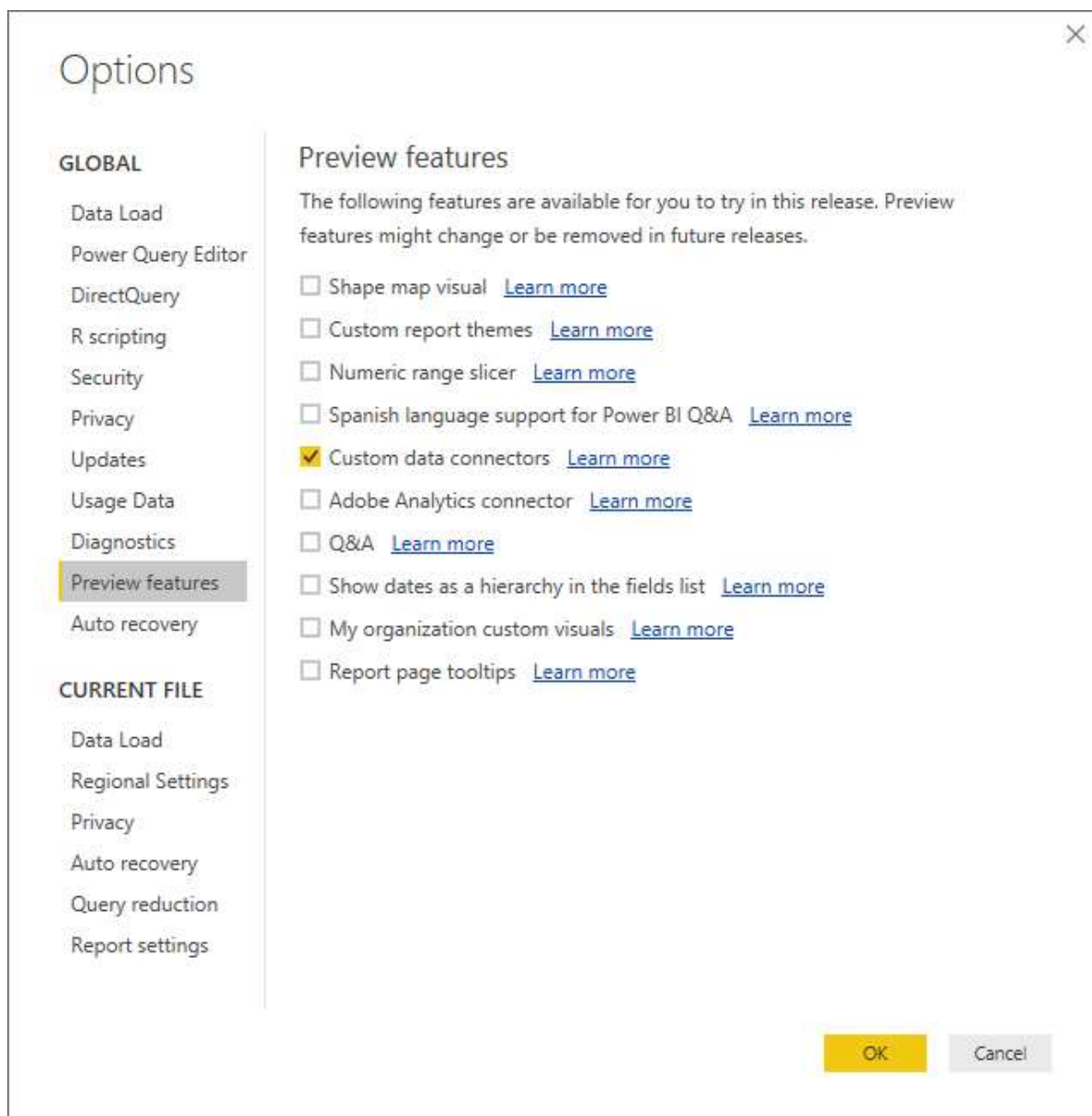
I adjusted QueryString just slightly, by changing *grant_type* to a fixed value of **authorization_code**, and I also need to send out the Client Secret as we're not using Basic authentication. This is documented in the Oura API [documentation](#).

Everything else in the HelloWorld sample I left untouched. In retrospect I only needed to do very small adjustments but the challenge is in finding out what to inject and where. I accidentally broke the Visual Studio project once and had to start all over. I'm thankful for [unlimited file versions](#) in Dropbox.

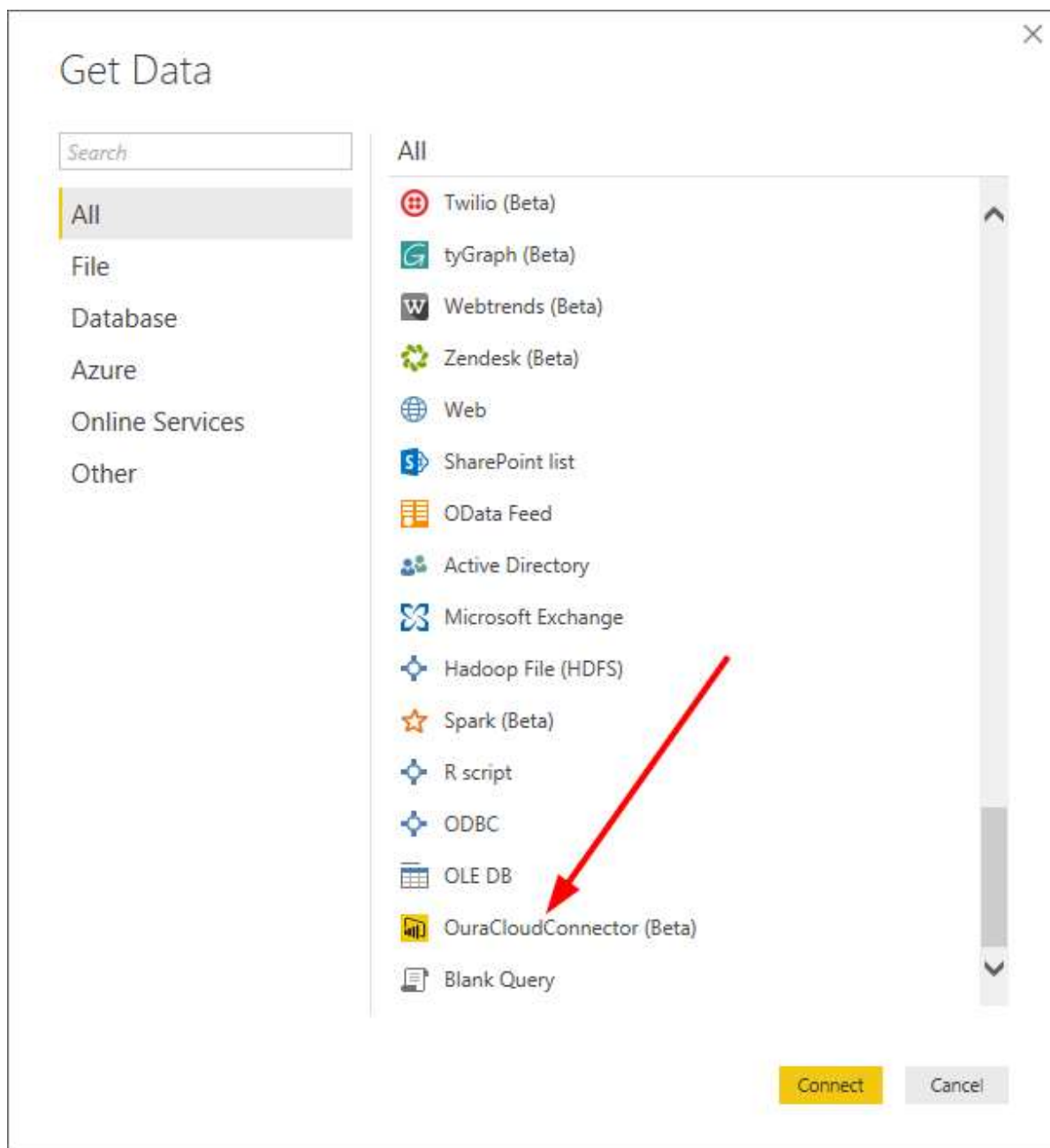
Putting it all together

Now that my code apparently compiles and builds it's time to run some tests! Upon compilation Power Query SDK produces a .MEZ file. Yet another new file format. But it's a .ZIP file so all is good in the world again. I found a [note](#) from Microsoft that in the future these files will be renamed to .PQX.

To deploy the custom connector, you need to copy it to *%USERPROFILE%\Documents\Power BI Desktop\Custom Connectors* and then restart Power BI Desktop. It picks up all custom connectors automatically, once you enable Custom data connectors in Preview features.



Selecting *Get Data* now produces the same long list of connectors but at the bottom you'll find your own custom connector! The '(beta)' tag is appended because within our class we set the property to True.



As we require a URL to query a prompt will ask for it first. I wanted to keep the custom connector as flexible as I could. This way, I can simply fiddle with the API without redeploying my connector.

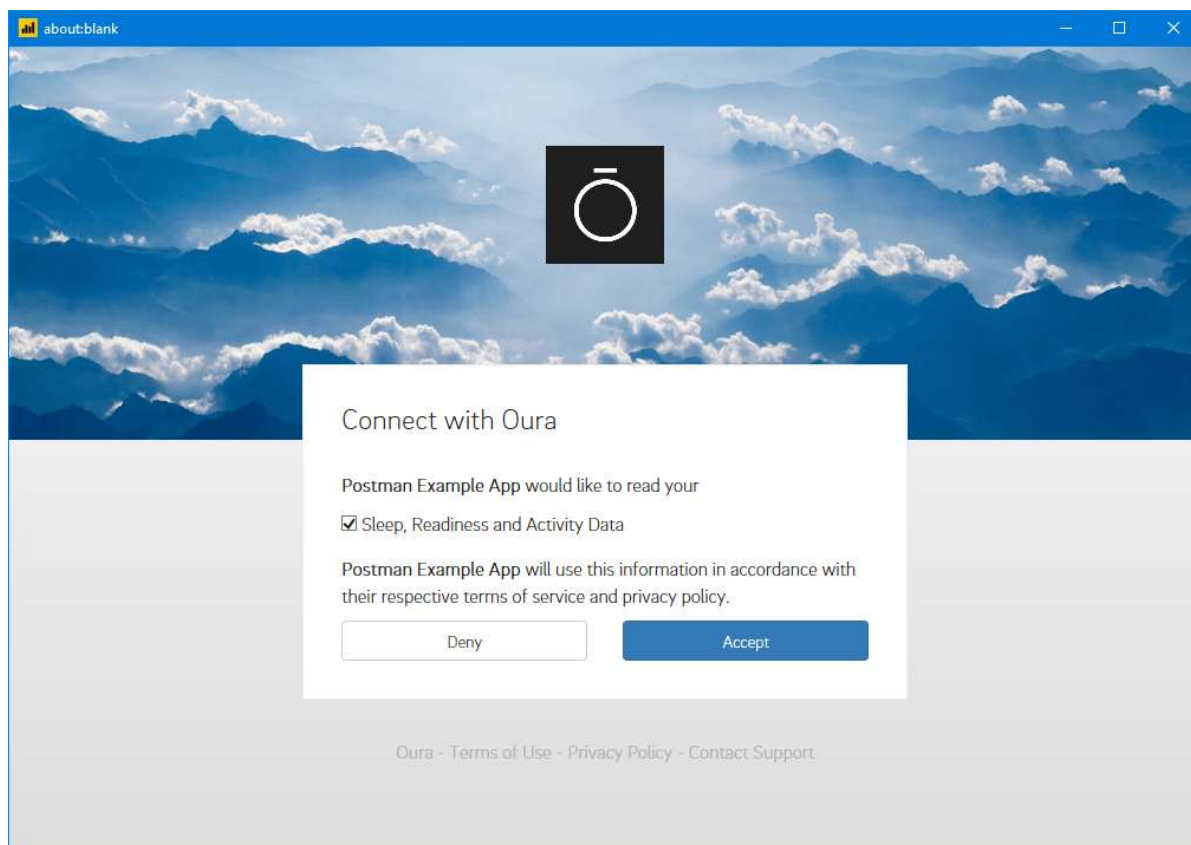


For URL I put <https://api.ouraring.com/v1/readiness?start=2019-02-24&end=2019-02-25>. This calls the Oura API for readiness data, between February 24th and 25th in 2019.

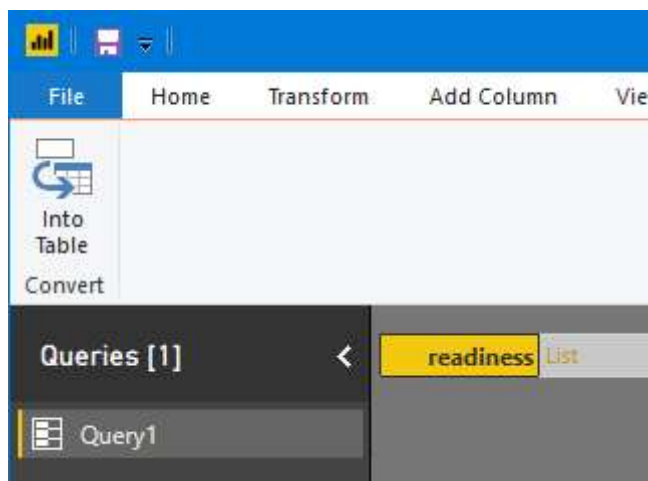
We then need to sign in and accept the license for Oura's API.



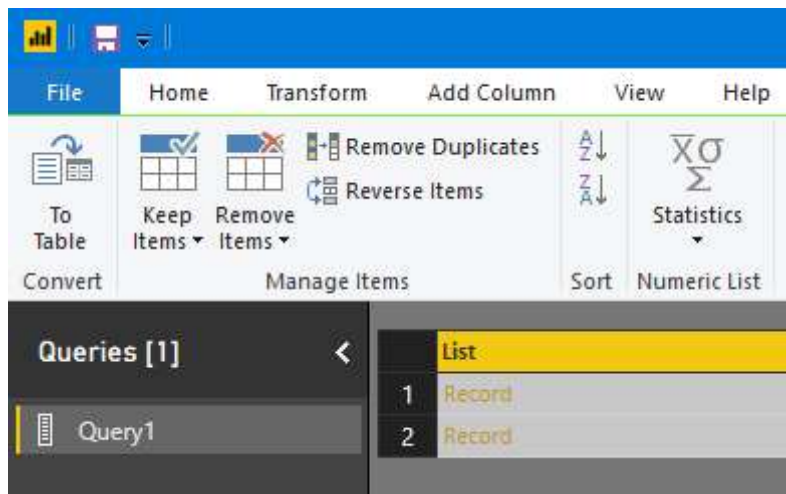
As we're using the Client ID and Client Secret (and the resulting access token) no actual user authentication needs to take place. Simply accept the Oura API terms of service and policy.



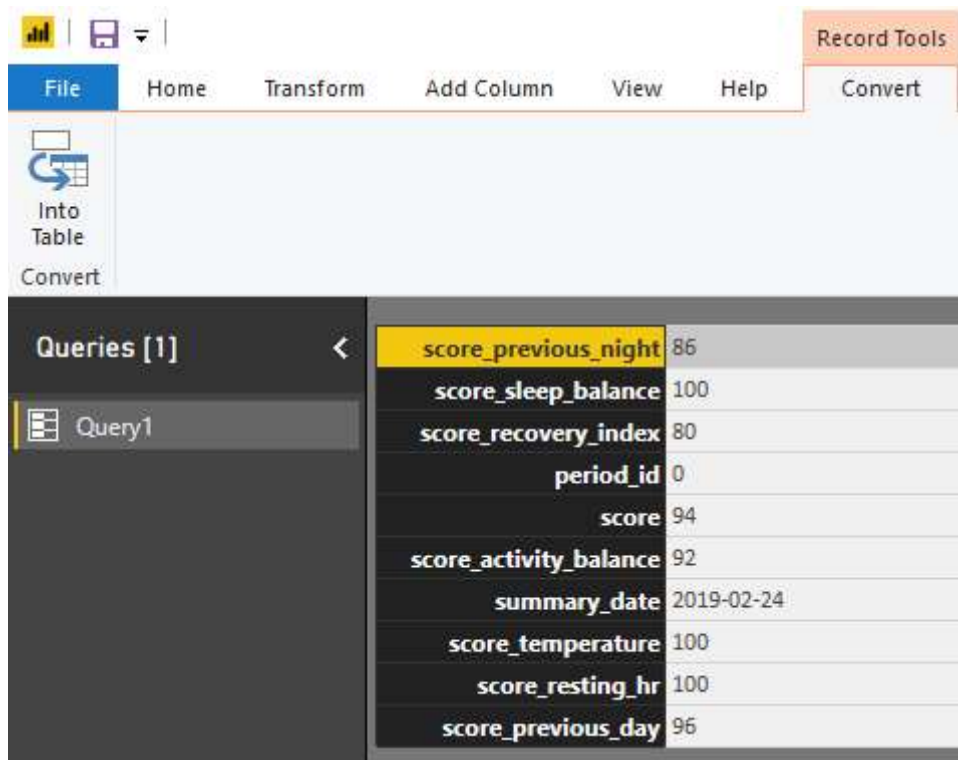
And next Power Query opens within Power BI Desktop allowing us to fiddle with the data.



We can see *readiness* as a list. There are two records, as we're querying for two dates.



Selecting the first record we finally see our data!



My readiness score for the night of 24th of February was an impressive 94 (out of 100). Anything above 85 % is considered a good recovery. I can now use Power Query's impressive roster of tools to transpose this table into a functional table and then automatically name the columns.

This table:

Untitled - Power Query Editor

File Home Transform Add Column View Help

Group By Use First Row as Headers Transpose Reverse Rows Count Rows

Table

Data Type: Text Replace Values Fill Pivot Column

Any Column

Queries [1]

Query1

	Name	Value
1	score_previous_night	86
2	score_sleep_balance	100
3	score_recovery_index	80
4	period_id	0
5	score	94
6	score_activity_balance	92
7	summary_date	2019-02-24
8	score_temperature	100
9	score_resting_hr	100
10	score_previous_day	96

Becomes this after transposing and using first row as headers:

	score_previous_night	score_sleep_balance	score_recovery_index	
1	86	100	80	

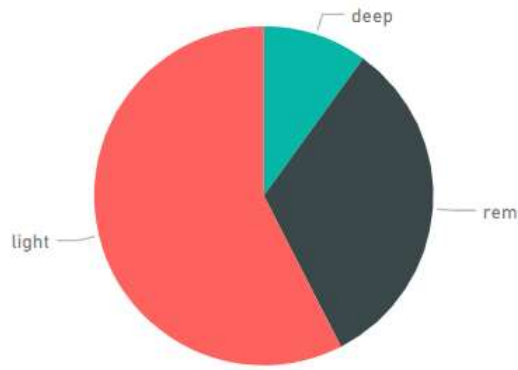
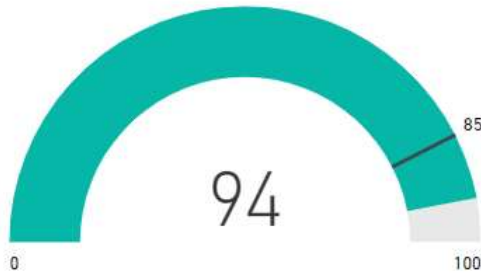
Once I close and apply I get all the data fields to work with:

VISUALIZATIONS	FIELDS
	<p>Search</p> <p>Query1</p> <ul style="list-style-type: none"> period_id score score_activity... score_previou... score_previou... score_recover... score_resting... score_sleep_b... score_temper... summary_date
<p>VALUES</p> <p>Drag data fields here</p>	
<p>FILTERS</p> <p>Page level filters</p>	

And from here's it's as simple as dragging and dropping fields and dimensions to my visualization!

score

deep, rem and light



In summary

This was, once again, a fun project to work on. I learned a lot from OAuth2 authorization, APIs in general, M Language and building custom connectors for Power BI. At times the challenges I had with M Language in general seemed quite frustrating but sleeping over it I was happy again to start troubleshooting the authorization issues.

The implementation is *far* from polished, but it works. I've published the solution in Github, and you're free to contribute or simply reuse my findings. See the repo [here](#).

Thanks for reading!



[Jussi Roine](#)

I work at Microsoft as Azure Developer Audience Lead. I'm a former Microsoft Most Valuable Professional & Microsoft Regional Director. Based in Helsinki, Finland.



Tags:

[API](#)

[Gadgets](#)

[Oura Ring](#)

[Power BI](#)

PREVIOUS

[Living with Oura Ring – the smart way to track wellness and activity](#)

NEXT

[Tracking, resolving, storing and presenting AKA.MS short links from social media using Serverless capabilities in Azure](#)

ALSO ON JUSSI ROINE

One password to rule them all: Migrating ...

a year ago • 7 comments

Fun with Azure Functions: Building ...

a year ago • 2 comments

Building a simple and secure DNS updater ...

a year ago • 2 comments

Getting started with Azure Database 1

3 months ago • 2 comments

I can't quite remember when I started using a password manager, or which ...

I sometimes find myself struggling with a technical issue that I just can't ...

Almost exactly a year ago I blogged about building a simple and secure DNS ...

I wanted to learn how to utilize Azure Database PostgreSQL, and write ...

What do you think?

11 Responses



Upvote

6



Funny

0



Love

3



Surprised

0



Angry

0



Sad

2

33 Comments

Jussi Roine



Disqus' Privacy Policy



Login



Recommend



Tweet



Share

Sort by Best

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS



Name



Rens Lokerse • 11 days ago

Hi Jussi, thanks a lot for your code, this helped me out big time! I am applying your code to connect with the ExactOnline API. I managed to login and retrieve the access token but when I try to query using the ExactCloudConnector.query.pq file I keep getting the error "We found extra characters at the end of JSON input". I verified the url I am using and know this is valid. Do you know how to solve this?

^ | v • Reply • Share ›



Rens Lokerse → Rens Lokerse • 10 days ago

Fixed it already, by adding a header.
source = Json.Document(Web.Contents(url, [
Headers = [
#"Accept" = "application/json;odata=verbose"
]))

^ | v • Reply • Share ›



AmitKumar • 12 days ago • edited

Hi sir,
Can we read read client_id, client_secret from parameter?

^ | v • Reply • Share ›



David Wuyts • a month ago

Hi Jussi, first of all thank you for this very extensive explanation of building an OAuth2 connector for PowerBI. I'm trying to customize your code to work with the following API: <https://www.peplink.com/ic2...> but for some reason I'm not able to get it to work. If I test the connector I always get the login page, but after login in nothing else is happening, and I get a "failed to set credentials" on the test Query Output. It would be awesome if you could push me in the right direction.

^ | v • Reply • Share ›



Jussi Roine Mod → David Wuyts • a month ago

Hi David, and thanks for reading! I'm not familiar with that specific API, but quickly looking at it, it seems to be a fairly standard one. Could you try accessing the API with just Postman, for example (see <https://www.postman.com/dow...> You could then easily verify whether your client ID actually works. If that passes, then try fiddling with your connector code by double-triple-checking all the URIs you're using. Hope this helps!

^ | v • Reply • Share ›



Thakshila • 2 months ago

I tried this connector in Power BI desktop and it works fine for me. Thanks for that. But when I try to add it into the power BI service I'm getting the following error. Do you have any idea about this. ? [View](#) —

uploads.disquscdn.com

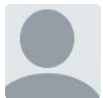
^ | v • Reply • Share ›



Jussi Roine Mod → Thakshila • a month ago

Hi Thakshila. Looking at the error, I feel you've omitted the text file(s) that include your Client ID and Client Secret for the Oura Cloud API. Perhaps verify that those are included when you compile the .PQ file.

^ | v • Reply • Share ›



Tal Aruety • 3 months ago

Great article! thanks

One question

The OAuth 2.0 REST API i work with has pagination

that is i can fetch up to 200 records per call

say i have 1000 records, i need to do an iteration which will pull the 1000 records in 5 calls (200 records per page, so page 1 , page 2 ,.....page 5) and have the iteration quit at the 6th iteration when the record set returned size is less then 200, thats how i know i reached the last page (in this example the 6th page will have 0 records)

And suggestion how to add that logic in M?

^ | v • Reply • Share ›



Jussi Roine Mod → Tal Aruety • 3 months ago

Hey Tall! Thanks for reading. Implementing pagination is a bit tricky, but thankfully someone already provided a solution for this. Check out this article: <https://datachant.com/2016/...>

In essence, you need to implement a function that tracks the current item, and provides pages for additional content and then loop through those.

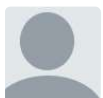
^ | v • Reply • Share ›



Tal Aruety → Jussi Roine • 3 months ago

Thanks man! that really helped, we created the connector we needed!

^ | v • Reply • Share ›



Adam Harris • 3 months ago

Hi [@Jussi Roine](#) ,

Thanks for sharing this! I'm following your instructions and trying to create a connector for NextRoll (<https://developers.adroll.c...>, but it keeps pinging me for login credentials when I try to run it. I noticed that the Oura documentation notes that you can include the secret in step 3 while the NextRoll documentation

the Oura documentation notes that you can include the secret in step 3 while the NextRoll documentation does not? I may just be thinking aloud through this comment here.

Anyways, thank you for writing this. It's gotten me closer to the solution than any other piece.

^ | v • Reply • Share ›



Jussi Roine Mod → Adam Harris • 3 months ago

Hey Adam! I had a quick look at the NextRoll API guidance. It says you can include the access token as a parameter, but usually you should be OK with using the access token via the Authorization header. Also make sure your scope is set to 'all'. Perhaps try using PostMan or cURL with an access token to see if you actually receive anything with the calls you're attempting to make. This should reveal any mistakes or missing parameters more easily!

^ | v • Reply • Share ›



Adam Harris → Jussi Roine • 3 months ago

Thanks Jussi! Postman has been great and I've been getting data through the connector, but I've been unable to refresh the connector without typing in my credentials again, so it looks like there's something off with my refresh flow. I'm getting the error that says the access token parameter can't be null. When you say I should be using the access token via the Authorization header, do you mean I should be including it like so?

 [View](#) — uploads.disquscdn.com

Thanks in advance!

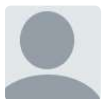
^ | v • Reply • Share ›




PunchcardCoder → Adam Harris • 3 months ago

The error is caused by having "authorization_code" as a fixed value in TokenMethod. This results in Refresh using "authorization_code" instead of "refresh_token". Just use variable grantType in TokenMethod instead and things work. "authorization_code" is only needed when FinishLogin calls TokenMethod.

^ | v • Reply • Share ›



vishnu reji • 9 months ago

 [View](#) — uploads.disquscdn.com I have tried this, working fine for me on desktop. thanks for this code. When I was trying this with power bi service I was facing a trouble. could you please help for me to find a solution for this.?

^ | v • Reply • Share ›



Jussi Roine Mod → vishnu reji • 9 months ago

Hi Vishnu. It appears you might have issues in 'procore-vishnu', which is probably the remote data source you are trying to connect. Could you check and verify the connection string that it works for you to access the remote system?

^ | v 1 • Reply • Share ›



REMI DESREUMAUX • a year ago

Very useful article! Thank you Jussi.

Is it possible to publish this report to a power BI server and share it with other end users of the Oura platform?

End users would provide their Oura credentials and visualize their own metrics straight from the browser.

^ | v • Reply • Share ›



Jussi Roine Mod → REMI DESREUMAUX • 3 months ago

Hey Remi, yep, this should work!

^ | v • Reply • Share ›



Koen van Besien • a year ago

Just wondering if this approach is considered safe?

As you are saving the client_secret with your connector in a text file?

^ | v • Reply • Share ›



Jussi Roine Mod → Koen van Besien • 9 months ago

It's 'safe' in the way that for private use, it's the easiest approach. In an enterprise environment you'd probably want to store the client secret some place secure, and retrieve it programmatically.

^ | v • Reply • Share ›



Fabien • a year ago

Hello Jussi,

Thank you very much for your blog, really helpful!

I managed to create the custom Connector, but now I want to have the client_id, client_secret and scopes in parameter of the query. But I can't find a way to drive these values from the "shared CustomConnector.Contents" to the "TokenMethod()".

Do you have any idea of how to do that?

Fabien

^ | v • Reply • Share ›



Rob van Zutphen • a year ago

Awesome post!! Thank you very much! I spent a lot of time searching how to create a custom connector, but as I'm not a developer your blog and githubdownload helped me out big time!

^ | v • Reply • Share ›



Jussi Roine Mod → Rob van Zutphen • 9 months ago

Great to hear, and thanks for reading!

^ | v • Reply • Share ›



Sam Woolerton • a year ago

Pro tip: you can actually get that data without a Custom Connector - just use the 'Web' data source and you skip all the hassles

^ | v • Reply • Share ›



Jussi Roine Mod → Sam Woolerton • a year ago

Hi Sam. Thanks for the tip. I'm not sure how that is supposed to work. Perhaps you've got a blog article outlining the steps or some other guidance on how to achieve this through the built-in tools against the Oura Ring API?

^ | v • Reply • Share ›



Sam Woolerton → Jussi Roine • a year ago

Honestly it's a pretty simple process when you try it

Get Data > Web > enter API URL > select Organisational account as auth method (like you currently do) > Bingo.

Will give you the same result with way less hassle - in writing the custom connector, you used Web behind the scenes (the Web.Contents function), so may as well save the time and use it directly

Another note - when you get to the final step, if you convert to a table (option in the ribbon, you can see it in your screenshots) instead of clicking into the record, you'll be able to track scores over time instead of just having the most recent value

^ | v • Reply • Share ›



Rob van Zutphen → Sam Woolerton • a year ago

I don't think this works straight from the "Get Data from Web", because of the Oath2 protocol. You'll need to GET and POST to exchange the tokens. The normal 'authentification-wizard' of the Web-connector doesn't do this.....

If I'm mistaken I'll gladly hear so because it would indeed save a lot of time and hassle!

1 ^ | v • Reply • Share ›



Sam Woolerton → Rob van Zutphen • a year ago

Are you sure? I've connected to quite a few sources that use OAuth2; you just select "Organisational account" as the authentication type and Power BI takes care of the tokens

^ | v • Reply • Share ›



Rob van Zutphen → Sam Woolerton • a year ago

Well.... not sure perse... but in my case, when I try to Sign in with 'Organizational account' I get the following errormessage:

"We were unable to connect because this credential type isn't supported for this resource. Please choose another credential type"

I tried all the other authentication options as well, but without any succes. With the custom connector I did get redirected to the loginpage, gave up the right permissions, and then I was good to go.

Maybe it depends on what kind of application you're (I'm) trying to access with Power BI?

Search ...



Follow @JussiRoine

About me





Hey there!

My name is [Jussi Roine](#). I'm based in Finland, and I work at Microsoft as an [Azure Developer Audience Lead](#). In practice, I talk and do Azure-related things.

I write about things that interest me, especially how I build solutions for myself and what I've learned over the course of my career so far.

You can find my contact info [here](#).

My Podcast





Perennial favorites

[Mastering Azure CLI](#)

[Turning on video during Microsoft Teams meetings – why, or why not?](#)

[Getting started with Windows Terminal](#)

[The comprehensive licensing guide to Microsoft Power Automate \(formerly Flow\) and Power Apps](#)

[Building a custom connector for Power BI that supports OAuth2 to visualize my wellness data](#)

Tags

[.NET Core API](#) [azure](#) [Azure CLI](#) [Build 2019](#) [Certification](#) [Cleaning up](#) [Collaboration](#) [community](#) [cost savings](#) [European](#) [SharePoint](#) [Conference](#) [event](#) [Events](#) [exam](#) [Gadgets](#) [Hackathon](#) [IoT](#) [Microsoft Flow](#) [Microsoft Ignite](#) [Migration](#) [News office](#) [365](#) [Oura Ring](#) [Power BI](#) [PowerShell](#) [Productivity](#) [reflection](#) [Security](#) [sharepoint](#) [SharePoint](#) [Saturday](#) [SQL Server](#) [Synology](#) [talks](#) [Teams](#) [Thoughts](#) [time](#) [Unifi](#) [Upgrade](#) [Windows 10](#) [Windows Hello](#) [Windows](#) [Terminal](#) [WordPress](#) [Working from home](#) [Working remotely](#) [writing](#)

Categories

[Building something](#) (37)

[Community](#) (19)

[Conceptual stuff](#) (7)

[Content and docs](#) (5)

[Events](#) (31)

[How to](#) (45)

[Microsoft Edge](#) (2)

[Podcast](#) (2)

[Productivity](#) (20)

[Review](#) (9)

[Skills](#) (2)

[Thoughts](#) (23)

[What's up](#) (15)

Made with ♥ in Helsinki by Jussi Roine