

1. Introduction

This document will guide you through using the Tempest2PBI custom Power BI connector to pull historical data from your tempest to use in Power BI reports.

2. Getting Started

To use the Tempest2PBI custom connector you will need the following things.

- The Power BI desktop application
- Your Tempest Weatherflow Device ID
- A tempest API Key

2. Downloading and installing Power Bi Desktop

- The Power BI desktop application is currently only available for Microsoft Windows and is available in the Microsoft Store. At the time of this writing the following link will take you to the Power BI Desktop application in the Microsoft store:

<https://aka.ms/pbidesktopstore>

Note: If you are a Fabric user and have an Apple Macintosh or Linux desktop you can still use this connector to connect to your Tempest Historical data using DataFlow Gen 2. Instructions for that are available in the Advance Usage section of this document.

3. Obtaining the API Key and obtaining your Device ID

- Go to the Tempest Weather App and sign in. <https://tempestwx.com>
- Click on settings in the upper right corner.
- Scroll to the bottom of the settings screen and select Data Authorizations and click on Create Token. This is the token you will use in the custom connector.
- To obtain the Device ID go to this link and replace [your_access_token] with the token you just acquired in the steps above:

[https://swd.weatherflow.com/swd/rest/stations?token=\[your_access_token\]](https://swd.weatherflow.com/swd/rest/stations?token=[your_access_token])

Congratulations, you have just made your first API call to your tempest stations data. You will see a long response, but your device ID is located near the top here:

```
{  
  "stations": [  

```

```
{
  "created_epoch": 1626015251,
  "devices": [
    {
      "device_id": 123456,
      "device_meta": {
        "agl": 1.8288,
        "environment": "indoor",
        "name": "HB-00032152",
        "wifi_network_name": ""
      }
    },

```

4. Configuring Days Back Parameters

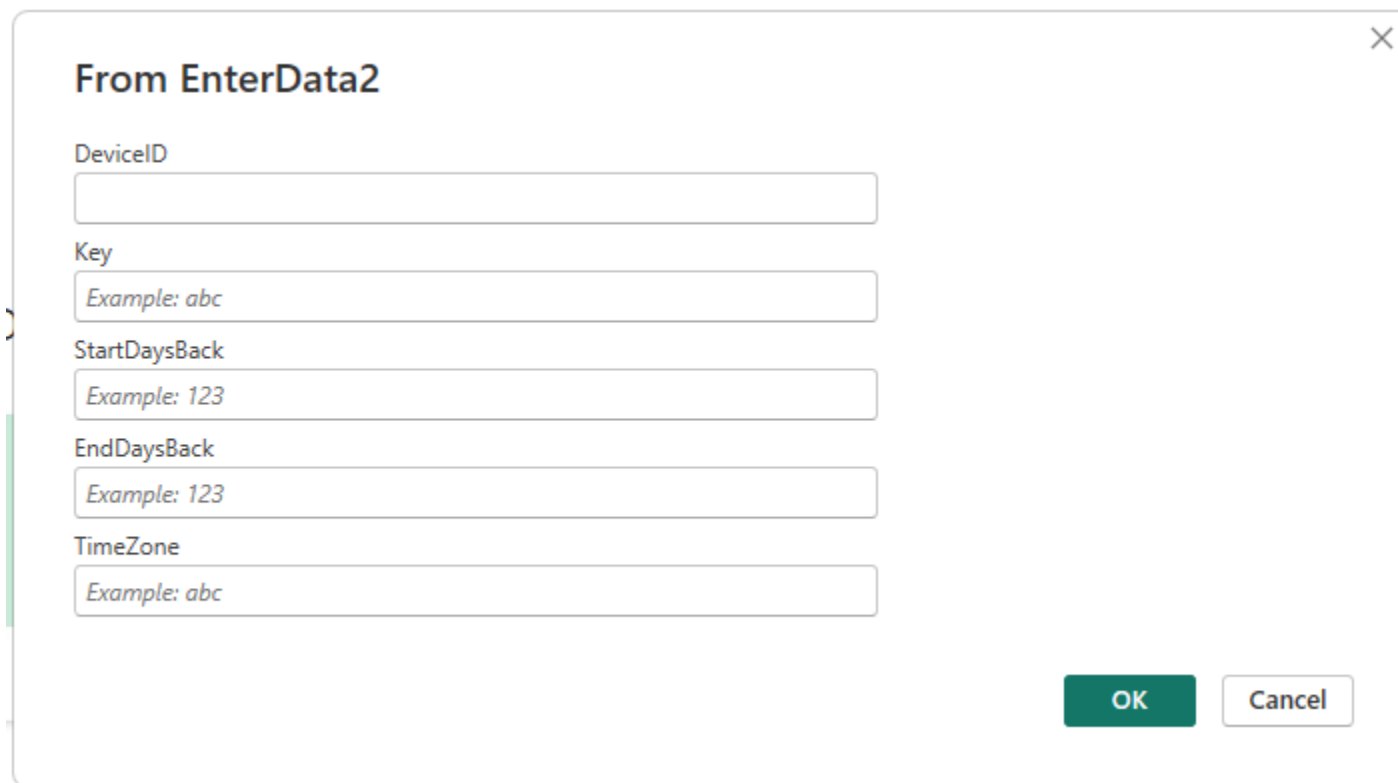
- You will be prompted for two days back parameters. Start Days Back and End Days Back. To pull history using the days back parameters, the API only supports pulling a single day at a time starting with today as 0. Yesterday is 1 the day before that is 2 etc.
- Since the API only allows pulling a single day at a time I create a table with a single column row containing the days back you want to pull. For example if you want to pull the last 7 days of data including todays data you would enter a Start Days Back of 0 and an End Days Back of 6. The connector will create a table that looks like this:

0
1
2
3
4
5
6

I will loop through this table and make a separate API call for each row (days back) in the table and put the results of all of those calls into a single table you can use for your report.

I have tested this connector up to 1000 days back and it work, but does take a few minutes to pull all of the data back since it is making 1000 API calls.

The prompts for the connector are as follows, enter your Device ID, API Key and start and stop days back in this dialog box.



The screenshot shows a dialog box titled "From EnterData2" with a close button (X) in the top right corner. It contains five input fields, each with a label and an example value:

- DeviceID**: An empty text input field.
- Key**: A text input field with the example value "Example: abc".
- StartDaysBack**: A text input field with the example value "Example: 123".
- EndDaysBack**: A text input field with the example value "Example: 123".
- TimeZone**: A text input field with the example value "Example: abc".

At the bottom right of the dialog box are two buttons: a green "OK" button and a white "Cancel" button with a grey border.

The TimeZone is to align the time in the report with your time.

Tempest historical data is stored in Unix epoch time and has a time zone of UTC. I convert from Unix time to DateTime and apply time zone math based on the TimeZone parameter in the above dialog box. If your time zone is west of UTC you will enter a negative number ex. -5, if your time zone is east of UTC you will enter a positive number and if your time zone is UTC enter a zero 0.

Note:

Tempest API documentation states that you can provide a date range to pull back all data within that date range, however I have not been able to get that to work. I have search online and found that others are having trouble with that method as well.

If anyone has an example of making a successful API call using a date range and is willing to share I will modify this connector to use that method as it would be a much cleaner approach.

5. Understanding the Resulting Table

- Data from the API is returned as metric values and the date is return as a Unix time stamp. I convert the epoch Unix date to a DateTime format with a time zone of UTC, you can modify the code in the connector to add or subtract your time from UTC. You can specify your time zone offset from UTC as a negative or positive number.
- I have converted the temperature from Celsius to Fahrenheit. If you use the metric system you can comment out the following line of code from the connector.
- `fahrenheitTable = Table.TransformColumns(dateTimeTable, {"temp", each _ * 9/5 + 32, type number})`
- I have converted rain accumulation reading from mm to inches.
- I have converted lightning distance from km to miles.
- You can comment out these conversion if you use the metric system. Comments in Power Query M code begin with //

The resulting table contains the following columns.

Tempest (type="obs_st")

Observation Layout

0 - Epoch (Seconds UTC) – [Converted to UTC Date Time](#)

1 - Wind Lull (m/s)

2 - Wind Avg (m/s)

3 - Wind Gust (m/s)

4 - Wind Direction (degrees)

5 - Wind Sample Interval (seconds)

6 - Pressure (MB)

7 - Air Temperature (C) – [Converted to Fahrenheit](#)

8 - Relative Humidity (%)

9 - Illuminance (lux)

10 - UV (index)

11 - Solar Radiation (W/m^2)

12 - Rain Accumulation (mm) - [Converted to inches](#)

13 - Precipitation Type (0 = none, 1 = rain, 2 = hail, 3 = rain + hail (experimental))

14 - Average Strike Distance (km) - [Converted to miles](#)

15 - Strike Count

16 - Battery (volts)

17 - Report Interval (minutes)

18 - Local Day Rain Accumulation (mm) - [Converted to inches](#)

19 - [NC Rain](#) Accumulation (mm) - [Converted to inches](#)

20 - Local Day [NC Rain](#) Accumulation (mm) - [Converted to inches](#)

21 - Precipitation Analysis Type (0 = none, 1 = Rain Check with user display on, 2 = Rain Check with user display off)

6. Advanced Usage

In this repository you have access to the source code for this connector. You can modify it to suit your needs.

If you are using a non windows machine to access Microsoft Fabric (Linux or a Mac) and would like to use this connector in a Dataflow Gen 2 to access your historical weather data you can do so by following these steps.

- Create a new Dataflow Gen 2
- Select Get data from another source ->
- Under New Sources select View more ->
- Search for 'Blank' and select Blank query
- Delete any text that is in the blank query and paste the following text in the blank query.
- You will need to add your DeviceID and Your Key as well as modify the daysList to include your start, end days back. Click Next to continue. You will be prompted to authenticate. Authenticate as anonymous.

```
let
    //DeviceID = "152639",
    //Key = "a307d230-391e-4b32-a807-80b15072982e",
    //TimeZone = "-5",
    //StartDaysBack = "0", //yesterday
    //EndDaysBack = "9", // the number of days of data you would like.
    The Values 0 through 9 will give you 10 days starting from yesterday

    timeZoneOffset = Number.FromText(TimeZone),
    sign = if timeZoneOffset < 0 then -1 else 1,
    // Create a list of days for which we want to fetch the data
    daysList = List.Numbers(StartDaysBack, EndDaysBack), // ** Change
    this line ** - the 0 is the start day and the 10 is the end day in this
    case I start with today (0) and go back 10 days (9)
    // Function to fetch data for a single day
    fetchDataForDay = (day as number) as table =>
        let
            // Make the API call for the current day
            apiResponse =
                Web.Contents("https://swd.weatherflow.com/swd/rest/observations/device/
                " & DeviceID & "?day_offset=" & Text.From(day) & "&token=" & Key),
            // Parse the JSON results
            parsedJson = Json.Document(apiResponse),
            // Check if 'obs' is null and process accordingly
```

```

•         fahrenheitTable = if parsedJson[obs] <> null then
•             let
•                 // Extract the JSONArray
•                 jsonArray = parsedJson[obs],
•                 // Convert the JSONArray into a table
•                 outputTable = Table.FromList(jsonArray,
Splitter.SplitByNothing(), null, null, ExtraValues.Error),
•                 recordsTable = Table.TransformColumns(outputTable,
{"Column1", each Record.FromList(_, {
•                     "epoch",
•                     "wind_lull",
•                     "wind_avg" ,
•                     "wind_gust",
•                     "wind_dir",
•                     "wind_smpl",
•                     "pressure",
•                     "temp",
•                     "humidity",
•                     "illuminance",
•                     "UV",
•                     "solar_radiation",
•                     "rain_acc",
•                     "precip_type",
•                     "avg_strike_dist",
•                     "strike_cnt",
•                     "battery",
•                     "rep_int",
•                     "day_rain",
•                     "nc_rain",
•                     "nc_day_rain",
•                     "precip_analysis"}), type record})),
•             expandedTable =
Table.ExpandRecordColumn(recordsTable, "Column1", {
•                 "epoch",
•                 "wind_lull",
•                 "wind_avg" ,
•                 "wind_gust",
•                 "wind_dir",
•                 "wind_smpl",
•                 "pressure",
•                 "temp",
•                 "humidity",
•                 "illuminance",
•                 "UV",
•                 "solar_radiation",

```

```

•         "rain_acc",
•         "precip_type",
•         "avg_strike_dist",
•         "strike_cnt",
•         "battery",
•         "rep_int",
•         "day_rain",
•         "nc_rain",
•         "nc_day_rain",
•         "precip_analysis"}),
•         // Convert epoch from a unix time stamp to DateTime
•         //dateTimeTable =
Table.TransformColumns(expandedTable, {"epoch", each #datetime(1970, 1,
1, 0, 0, 0) + #duration(0, 0, 0, _), type datetime}},
•         dateTimeTable =
Table.TransformColumns(expandedTable, {"epoch", each #datetime(1970, 1,
1, 0, 0, 0) + #duration(0, 0, 0, _) - #duration(0, sign *
timeZoneOffset, 0, 0), type datetime}},
•         // Convert temp from C to F.
•         fahrenheitTable =
Table.TransformColumns(dateTimeTable, {"temp", each _ * 9/5 + 32, type
number}},
•         // Convert rain_acc from mm to inches
•         rainAccInchesTable =
Table.TransformColumns(fahrenheitTable, {"rain_acc", each _ *
0.0393701, type number}},
•         // Convert avg_strike_dist from km to miles
•         avgStrikeDistMilesTable =
Table.TransformColumns(rainAccInchesTable, {"avg_strike_dist", each _ *
0.621371, type number}},
•         // Convert day_rain from mm to inches
•         dayRainInchesTable =
Table.TransformColumns(avgStrikeDistMilesTable, {"day_rain", each _ *
0.0393701, type number}},
•         // Convert nc_rain from mm to inches
•         ncRainInchesTable =
Table.TransformColumns(dayRainInchesTable, {"nc_rain", each _ *
0.0393701, type number}},
•         // Convert nc_day_rain from mm to inches
•         ncDayRainInchesTable =
Table.TransformColumns(ncRainInchesTable, {"nc_day_rain", each _ *
0.0393701, type number})
•         in
•         ncDayRainInchesTable
•         else

```

```

•         #table(type table [epoch=datetime, wind_lull=number,
wind_avg=number, wind_gust=number, wind_dir=number, wind_smpl=number,
pressure=number, temp=number, humidity=number, illuminance=number,
UV=number, solar_radiation=number, rain_acc=number,
precip_type=number, avg_strike_dist=number, strike_cnt=number,
battery=number, rep_int=number, day_rain=number, nc_rain=number,
nc_day_rain=number, precip_analysis=number], {})
•         in
•         fahrenheitTable,
•         // Fetch data for all days and combine into a single table
•         allData = Table.Combine(List.Transform(daysList, each
fetchDataForDay(_)))
•         in
•         allData
•

```

7. Troubleshooting

- Days are missing from my table: If your tempest goes off line, it will stop sending data to the backend service and you will have no historical weather data for the time that the Tempest was off line. If your tempest is off line for more than a day, the API call for that day will succeed, but the Json array that contains historical data will be null. If this connector encounters a null array, it will ignore it and keep processing. The result is that you will have missing days in your data.
- The Weatherflow API returns all values in metric and I have converted those to imperial. If you prefer metric values you can comment out the lines that do the conversion.