PI World 2019 Lab

Programmability in the PI Connector for UFL

OSIsoft, LLC
1600 Alvarado Street
San Leandro, CA 94577 USA
Tel: (01) 510-297-5800
Web: http://www.osisoft.com

Published: March 20, 2019

Table of Contents

1. Introduction

1.1 Overview

This lab will explore the new features of the PI Connector for UFL to parse multi-column data files and using RESTful data sources with the PI Connector for UFL to collect information from a public REST API, and then send the data to a PI Server (Data Archive and AF server.)

In this lab, users will gain the knowledge required to perform basic setup, configuration and troubleshooting for this connector to be able to parse data from REST APIs and data files.

1.2 Setup

This lab assumes the following:

- Running instances of the PI Server (Data Archive, AF Server) and PI Vision
- Running instance of a PI Connector for UFL

2. Background

2.1 What is the PI Connector for UFL

The PI Connector for UFL allows the transfer of contextual and time-series data from ASCII files, REST clients and REST servers to the OSIsoft PI Server (Data Archive, Asset Framework).

2.2 Terminology and Important Concepts

2.2.1 REST

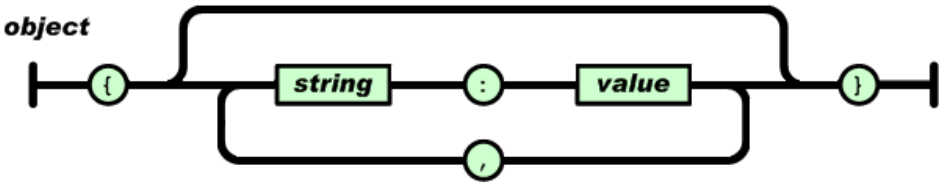
- REpresentational State Transfer is an architectural style for providing standards between RESTful systems. There are 3 HTTP verbs (kind of operation) that can be used with the PI Connector for UFL to interact between REST server and a REST client:
    - PUT – updates the specific resource
    - POST – creates a new resource
    - GET – retrieves a specific resource

2.2.2 HTTPS

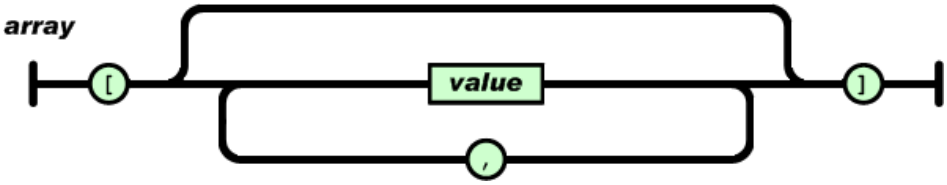
- Encrypted adaptation of HTTP for secure communication across a network.

2.2.3 JSON

- JavaScript Object Notation (JSON) is a lightweight data-interchange format that is easy for both humans and machines to read. It has widely replaced XML since it is easier to read and parse.
- JSON is built on two structures:
    - Object: collection of string/value pairs
    - Array: collection of values
- {Object} – **unordered** data held in curly braces

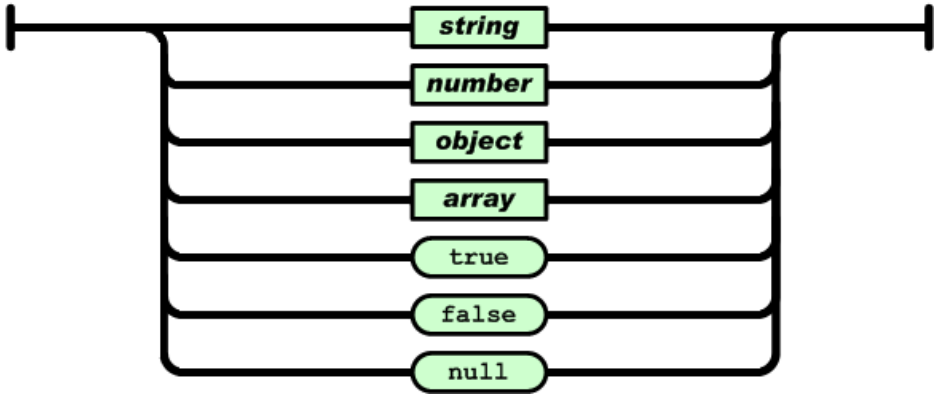


- [Array] – **ordered** list of objects held in square brackets



- Value:

value



A JSON file will be sent between the server and the client. Here is an example JSON object in its compact form:

```
{"list":[{"weather":{"Temperature":20,"Condition":"clear sky"},"name":"San Francisco"},{"weather":{"Temperature":8,"Condition":"mist"},"name":"London"}]}
```

For readability's sake, the JSON object is generally represented as such:

```
{
    "list":[
      {
        "weather":{
          "Temperature":20,
          "Condition":"clear sky"
        },
        "name":"San Francisco"
      },
      {
        "weather":{
          "Temperature":8,
          "Condition":"mist"
        },
        "name":"London"
      }
    ]
}
```

Object 1

weather\temperature

weather\condition

Object 2

2.3 REST API

An **API** is an application programming interface. It is a set of rules that programs use to communicate with each other. An API server is developed to allow clients to communicate with it. **REST** determines the architecture of the API. Each URL is called a **request** while the data sent back is called a **response**. As part of this lab, Exercise 2 uses the REST API from OpenWeatherMap. The API documentation contains the parameters supported, examples, and more.

For a reference list of available public REST APIs, ProgrammableWeb has a list of over 20,000 available REST APIs.

2.4 Data Flow

The PI Connector for UFL allows three **Data Source Types**:

- Files

  From a windows directory – the files can be of any extension, as long as they are encoded using a supported encoding.
- REST Client

  The PI Connector acts as a REST client and executes GET requests on the REST endpoint specified. The returning data format can be JSON, XML, CSV, or plain text.

Note:
For HTTP servers requiring parameters in the HTTP header, they can be defined manually in the "%pihome64%\Connectors\UFL\Configuration\Datasource.config.json"

- REST Server

The following figure displays the data flow for the PI Connector for UFL and typical data sources.

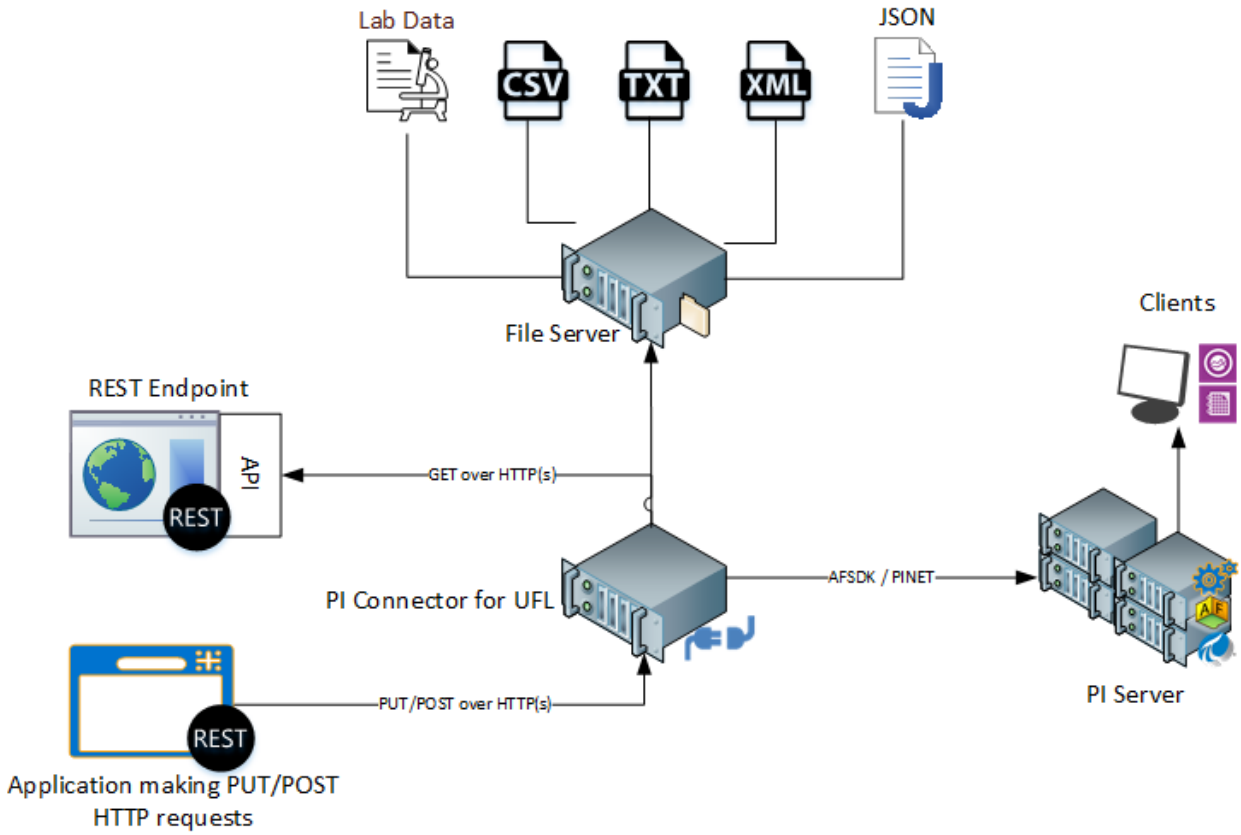


Figure 1 - PI Connector for UFL Data Flow

2.5 PI Connector for UFL vs PI Interface for UFL

The following table summarizes the main differences between the PI Interface for UFL and the PI Connector for UFL. The following lab focuses on the PI Connector for UFL and the exercises in the workbook will showcase the native JSON parsing, CSV looping, REST client functionality, and AF integration (AF Element, AF Attribute, and AF Template creation).

Table 1 - Differences between the PI Interface and the PI Connector

	PI CONNECTOR FOR UFL	PI INTERFACE FOR UFL
DATA SOURCE TYPE	Files	Files
	REST Client	Serial
	REST Server	POP3 (emails)
AF SUPPORT	Yes	No
PI POINT CREATION	Yes	Yes
PI POINT CONFIGURATION	No	Yes
NATIVE JSON PARSING	Yes	No
SUPPORTED ENCODING	Extended ASCII	Extended ASCII
	Unicode (UTF-8)	
LOOPING FUNCTIONS FOR CSV FILES	Yes	No
WRITING TO FUTURE PI POINT	Yes	Yes
FUTURE PI POINT CREATION	No	Yes
CUSTOMIZABLE COMPRESSION SETTINGS	Yes	Yes
BUFFERING MECHANISM	Embedded	PI Buffer Subsystem
ENCRYPTED COMMUNICATION TO DATA ARCHIVE (WIS)	Yes	Only with PI API 2.x

2.6 Tools and Utilities

Throughout this lab, a number of tools and utilities are used:

- PI Connector Administration website – All of the configuration for a PI Connector is performed through this web app: data source configuration, the destination PI Data Archives and PI AF Servers, etc. Additionally, this web app also allows the connector to be stopped/started and provides diagnostic information.
- PI System Explorer – This is the primary client tool for PI AF. This tool allows a user to interact with one or more PI AF servers: the hierarchy can be browsed and manipulated, the metadata can be explored, attributes and elements can be configured, and so on.

- PI System Management Tools – With this management utility, a user can connect to one or more PI Data Archives to create/edit PI points, explore archive and snapshot data, view and change PI Data Archive settings, and other related tasks.

3. Data Parsing in the PI Connector for UFL

Regardless of the source, incoming data is treated as a set of consistently formatted lines, which are referred to as messages. The PI Connector for UFL uses and INI file to parse the data.

Note:
The maximum line length supported by PI Connector for UFL is 5120 characters.

3.1 Basic INI configuration file structure

The INI configuration file is formatted as follows:

[FIELD]

Defines and declares data types for the individual fields that receive data.

[MSG]

Defines the types of incoming messages, and assigns a name that is used to define the section where the message is divided.

Per message sections

For each message that is defined in the MSG section, the *Per message sections* filter incoming messages, divide the messages into fields, process the fields, and then write the results to PI tags, PI AF elements, or event frames. These sections can contain processing logic, such as logic that redirects to other sections and skips lines from the input stream.

3.2 Collections

A variable of the *Collection* data type can accommodate any of the supported data types (DateTime, Time, String, Int32, Number). It is an array of other variables. In practice, *Collections* are most commonly used to stored PI Point names, AF Attributes names (dynamic and static), values, and timestamps. Sending a command to PI to create PI Points simply becomes one operation executed by passing an array of PIPoints, Timestamp(s), and Values.

Values of name-value pairs can be added to the *Collection* array using the `Add()` function:

```
Collection = Add([name,] value)
```

3.3 Predefined Variables

__MESSAGE

The content of the current message (line).

__ITEM

A string variable which is assigned a value each time the `JsonGetItem()` or `CsvGetItem()` functions are evaluated.

__ITEM_Name

A string variable which is assigned the name of the selected JSON element each time the `JsonGetItem()` function is evaluated.

3.4 Main Functions to Interact with the PI Server

3.4.1 Sending data to PI Point(s)

Note:
If the PI Point, AF Element, AF Attribute, or AF Template does not exist, the connector will create it.

Note:
The square brackets indicate that those parameters are optional and can be omitted.

```
StoreInPi( Tag, ElementAttribute, Timestamp, Value[, Status, Questionable] )
StoreEvent( Tag, ElementAttribute, Timestamp, Value[, Status, Questionable] )
```

PARAMETER	DESCRIPTION	DATA TYPE
TAG	Target PI Point	`String`
ELEMENTATTRIBUTE (OPTIONAL)	Attribute Name of the corresponding AF Element. If omitted, the point cannot be referenced through an AF Element	`String`
TIMESTAMP (OPTIONAL)	Timestamp to be recorded with the value. If omitted, current system time is recorded	`DateTime`
VALUE	Value	`String, Int32, Number, DateTime`

```
StoreEvents( TagNames, ElementAttributes, TimeStamp(s), Values[, Statuses, Questionables] )
```

PARAMETER	DESCRIPTION	DATA TYPE
TAGNAMES	Target PI Points	Collection of `String`
ELEMENTATTRIBUTES (OPTIONAL)	Attribute Names of the corresponding AF Element. If omitted, the point cannot be referenced through an AF Element	Collection of `String`
TIMESTAMP(S) (OPTIONAL)	Timestamp to be recorded with the value. If only one timestamp is supplied, it will be used for all values. If omitted, current system time is recorded	`DateTime` or Collection of `DateTime`
VALUES	Values	Collection of: `String, Int32, Number, DateTime`

3.4.2 Create or Update AF Element

```
StoreElement( Path, Template[, DynAttributes, StatAttributes] )
```

PARAMETER	DESCRIPTION	DATA TYPE
PATH	Back slash delimited path to an AF Element. Example: `Gateway\Device\Dataset`	`String`
TEMPLATE	Template name in AF for the AF Element	`String`
DYNAMIC ATTRIBUTES (OPTIONAL)	Collection of PI Point name (`TagNames`). The AF Attributes will be named based on the value of `ElementAttribute`(s) passed in the `StoreEvent(s)()` function from which the value was recorded.	Collection
STATIC ATTRIBUTES (OPTIONAL)	Collection of static attributes. These attributes will be created with a DataReference = None	Collection

3.5 Native Functions for CSV & JSON

3.5.1 FOREACH()

With the `FOREACH()` code flow control, the connector can loop through sets of items in the following well-known data formats: JSON and CSV.

```
FOREACH(condition) DO expression(s) ENDFOR
```

The `condition` can only include one of the following functions:

- `CSVGetItem()`
- `JSONGetItem()`

Below is a simplified example explaining the principle. It iterates through a collection of comma-separated items and add the individual items to a variable of the `Collection` data type. This allows the `Collection` *Values* to be populated with all the items present in the line being parsed. This syntax is more flexible as it easily allows iterating through a variable number of items (columns for CSV)

```
FOREACH(CsvGetItem(__MESSAGE,",")) DO
      Values = Add(__ITEM)
ENDFOR
```

3.5.2 CsvGetItem()

```
CsvGetItem("Csv_input", "Delimiter")
```

This function applicable in the condition part of the `FOREACH()` statement. It populates the predefined string variable `__ITEM` with the item that is present between the `Delimiter`.

- `Csv_input(String)` is a succession of delimited values
- `Delimiter(String)`can be one or more characters – case sensitive

For example, a CSV using commas to separate values can be iterated in a FOREACH() loop using the example above.

3.5.3 JsonGetItem()

```
JsonGetItem("Json_input", "Selector")
```

This function applicable in the condition part of the FOREACH() statement. It populates the predefined string variables __ITEM and __ITEM_NAME with the object selected.

- Json_input(String) is a valid JSON array
- Selector(String) can be one or more characters – case sensitive

3.5.4 JsonGetValue()

```
JsonGetValue("Json_input", "Selector")
```

As the name indicates, JSONGetValue() is used to obtain a value from a string/value pair within an object.

The following example would retrieve the value of Weather\Temperature. As part of a FOREACH() statement, it would then iterate for each Objects in the array:

```
Temperature_Number = JsonGetValue(__ITEM,"Weather\Temperature")
```

4. Exercise 1 – Parsing a CSV File with Variable Number of Columns

This exercise explores the design of the configuration file (.ini file) to parse a typical text file. The learning environment provided has a freshly installed connector, and now only requires basic configuration to begin data collection.

4.1 Objectives

- Create a data source for the connector
- Configure .ini file for typical dataset format

4.2 Step 1: Add a PI Data Archive and PI Asset Framework Server

Add and configure PI Data Archive and PI AF for communication with the connector.

Note:
Modifications can be completed without stopping the connector data collection.

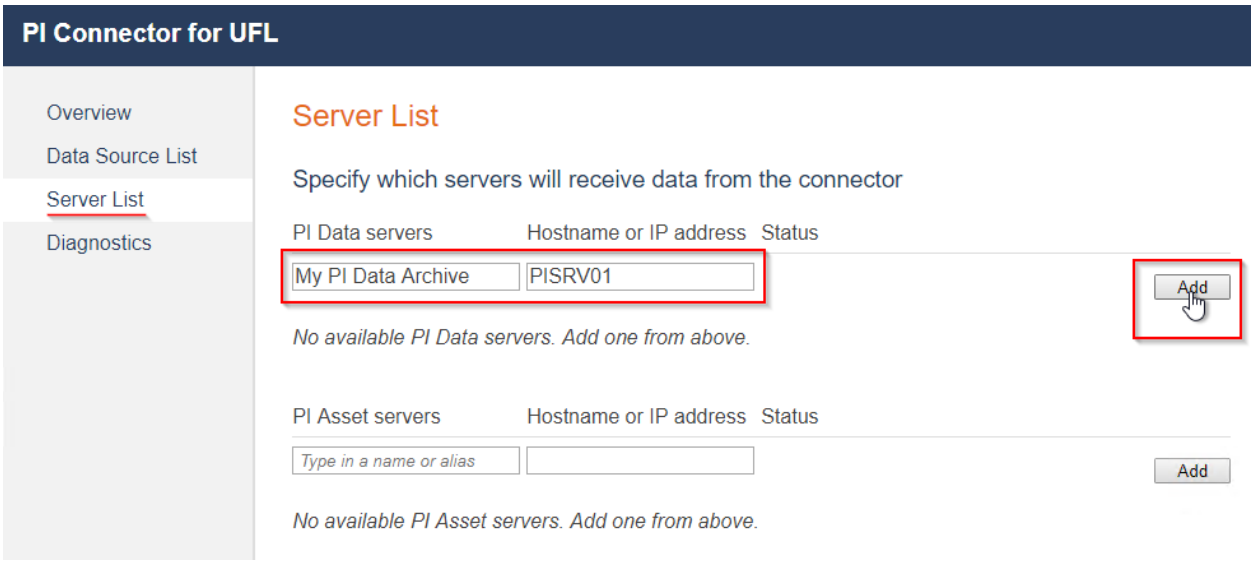
1. Open the Administration page for the UFL Connector. There is a shortcut on the desktop as well as the start menu under **PI System > PI Connector for UFL Administration**.

2. If prompted for credentials, use:
    a. Username: pischool\student01
    b. Password: *provided in class*

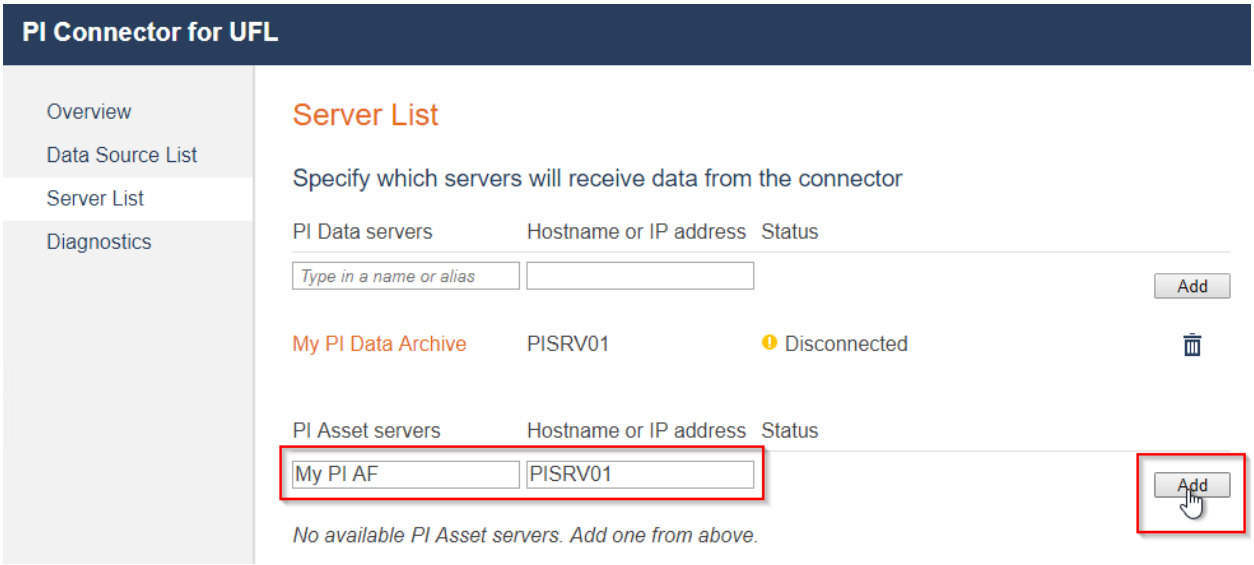
3. On the PI Connector Administration page, click the **Server List** link on the left side of the screen. PI Data servers lists the PI Data Archives the connector will send data. The PI Asset servers lists the PI Asset Framework server that will store the logical hierarchy of elements and metadata.

4. To add a PI Data Archive, a name and hostname is required. A name can be any identifier used to describe the PI Data Archive, including the machine name. The hostname is the network information of the PI Data Archive.

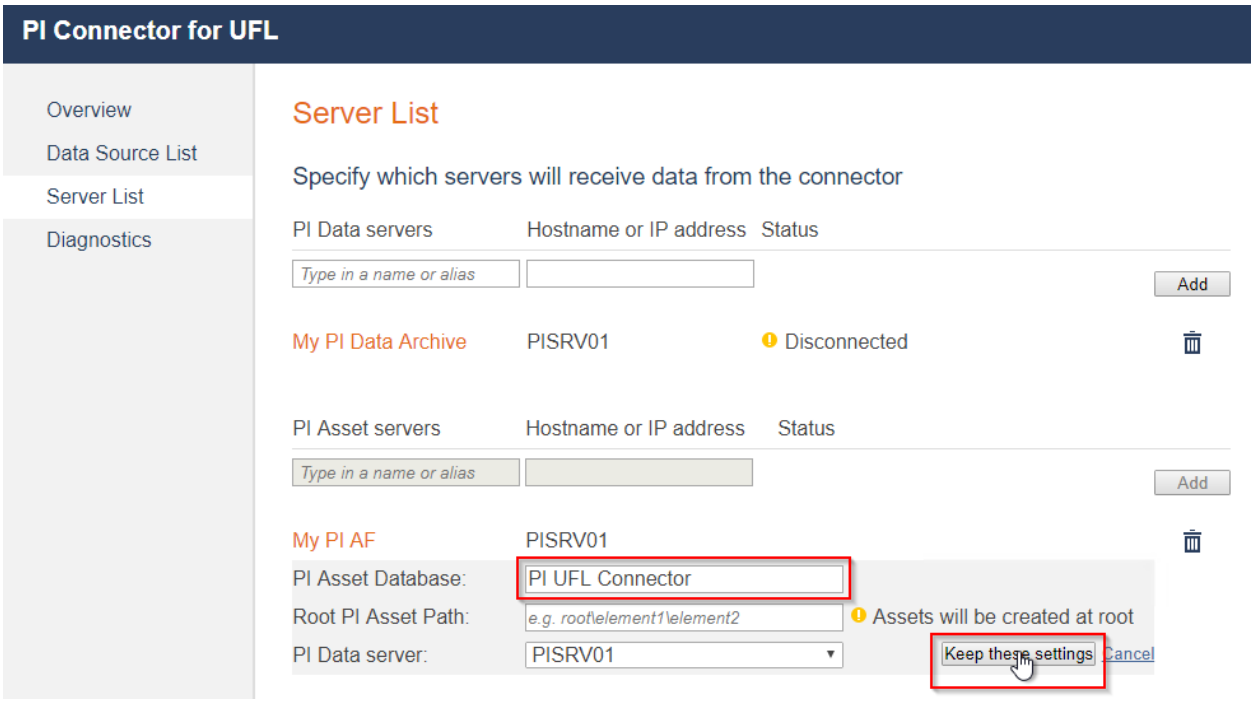
5. Add a name for the PI Data Archive and use **PISRV01** for the hostname. Select **Add** to save changes.



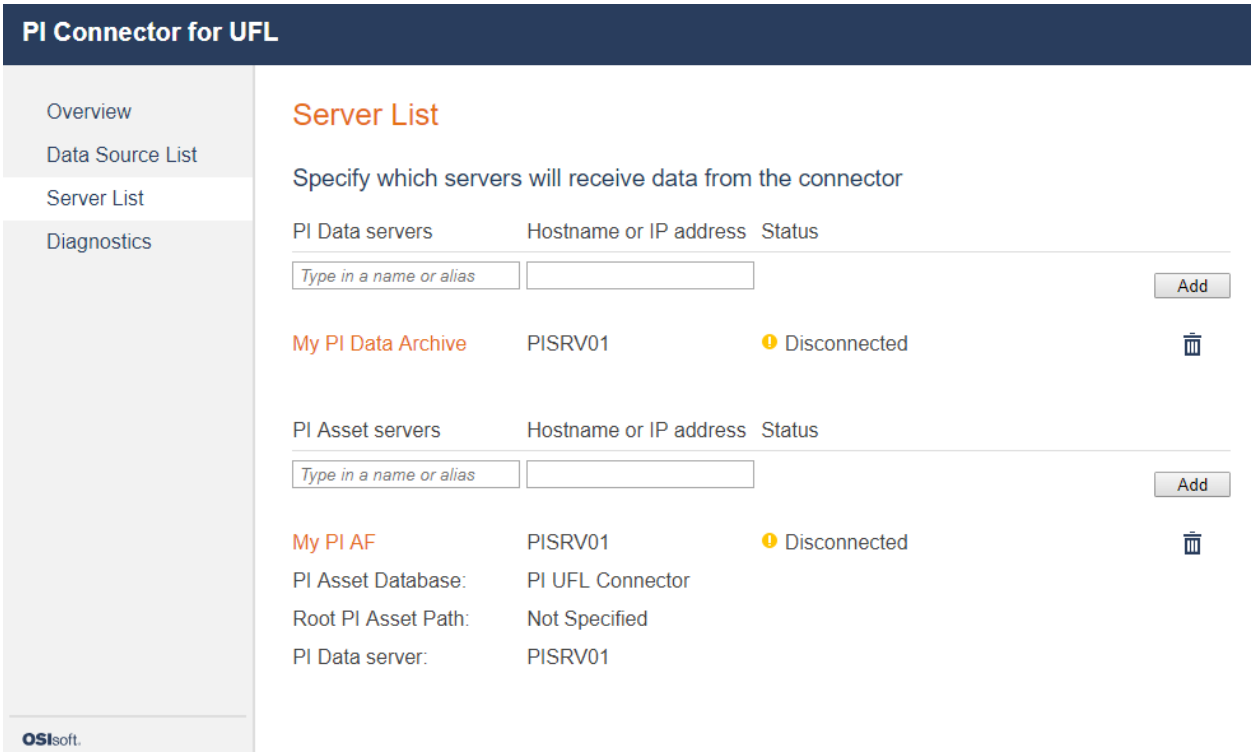
6. Perform the same for the PI Asset servers section. Select a name for the PI AF server and use **PISRV01** for the hostname. Select **Add** to save changes.



7. After adding the server, specify the name of the AF Database to be created, which will be **PI UFL Connector**. The other items can be left as is. Select **Keep these settings** to save changes.



8. After adding the servers and saving changes, the final list should look like the following:



4.3 Step 2: Configure data sources

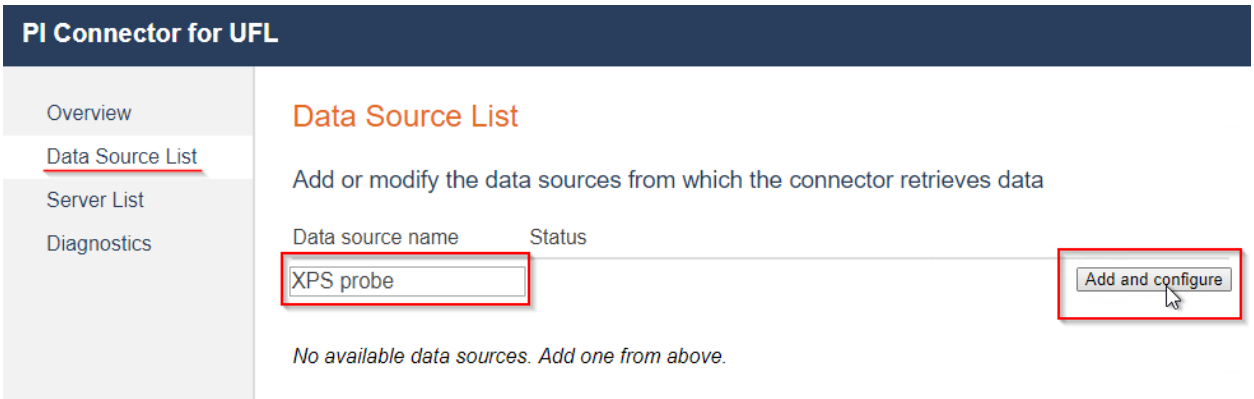
The connector will now parse any incoming data and send it to the specified PI Data Archive and PI Asset Framework servers specified.

In this exercise, data from X-ray photoelectron spectroscopy (XPS) probe. A sample is loaded into a machine, and the analyte's elemental composition is measured using X-rays. This measurement data generated is presented in a .csv file in the following format:

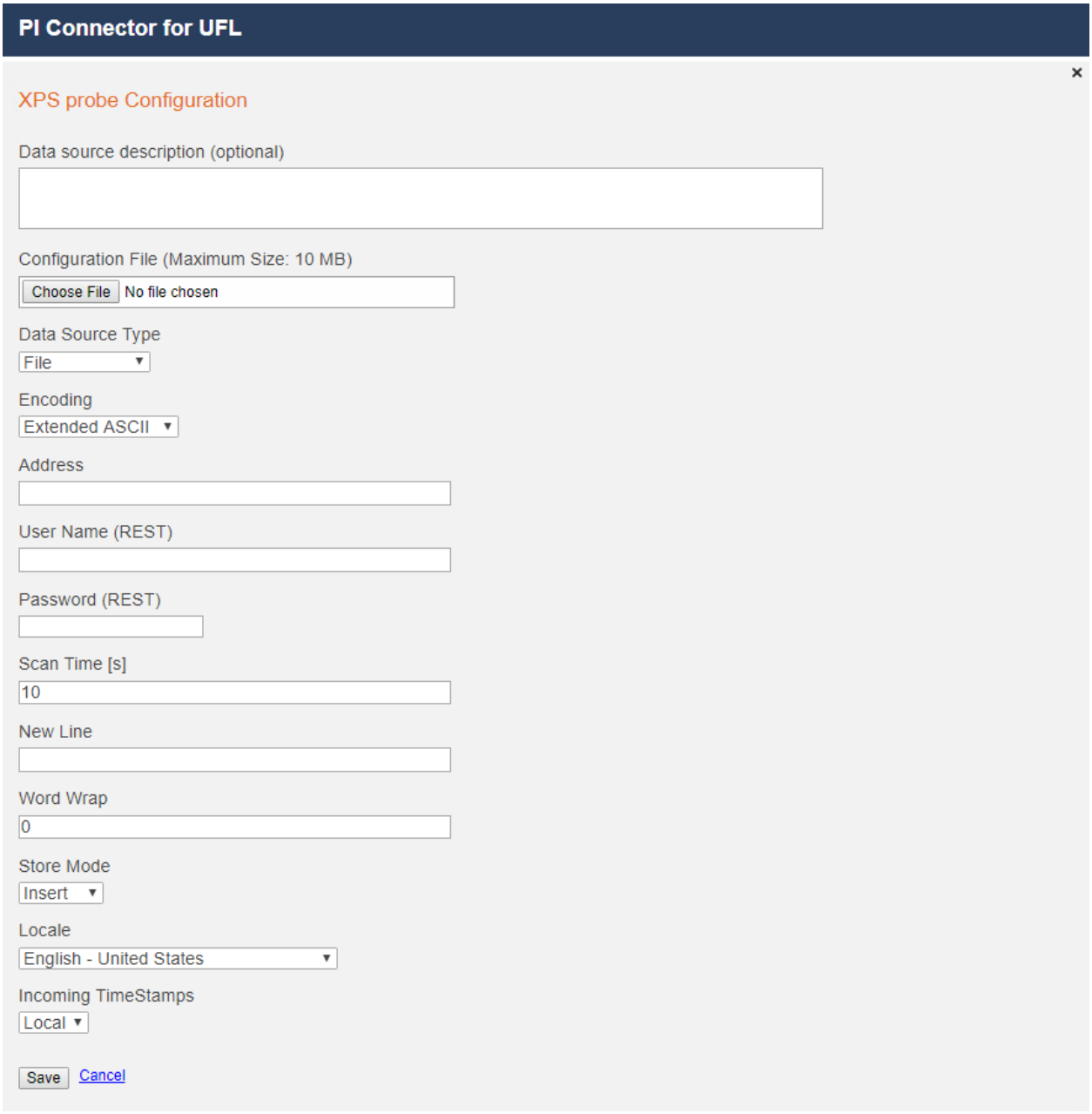
```
TimeStamp,Hydrogen,Helium,Lithium
1/27/2019 06:10,20.34954013,23.8979401,44.03301038
1/27/2019 06:20,52.25473449,55.95751127,19.57478451
1/27/2019 06:30,35.3680037,83.62752702,80.83267956
1/27/2019 06:40,67.87165734,12.05941163,46.46192035
1/27/2019 06:50,87.26456026,21.19993289,69.29936443
1/27/2019 07:00,73.3539596,14.81267681,40.92914766
1/27/2019 07:10,8.475626883,43.57027272,32.96403702
1/27/2019 07:20,22.36870781,37.73167597,34.7768616
1/27/2019 07:30,40.14540103,54.45435054,54.61024746
1/27/2019 07:40,61.94443169,84.13089856,15.19648365
1/27/2019 07:50,91.6951069,47.78477007,60.06808156
1/27/2019 08:00,35.8142939,19.5996841,66.49259103
1/27/2019 08:10,95.17955006,92.30584051,87.95001521
1/27/2019 08:20,29.12528214,3.817737961,70.30317159
1/27/2019 08:30,85.28168446,41.99492191,14.75110356
1/27/2019 08:40,98.69864031,98.2288604,28.03794947
1/27/2019 08:50,90.66766549,5.156088295,54.53605229
1/27/2019 09:00,80.16275714,0.728795825,85.63577164
1/27/2019 09:10,38.71803075,80.03045369,80.82624252
1/27/2019 09:20,93.80282208,80.68784985,53.29287652
```

The first row is the header row, labeling each column in the subsequent lines. The timestamp is designated in the first column, and the remaining columns are the measurement values for each analyte. Depending on the analyte's composition, the number of columns will vary. In the PI Interface for URL, the accommodating logic to parse a file with variable number of columns is cumbersome. The simplicity and flexibility of new functions in the PI UFL Connector are demonstrated in this example.

1. To configure this data source, select **Data Source List** on the left side of the connector UI. In the Data source name field, type **XPS probe** and click **Add and configure**.



2. An empty data source configuration page will load, and should look like

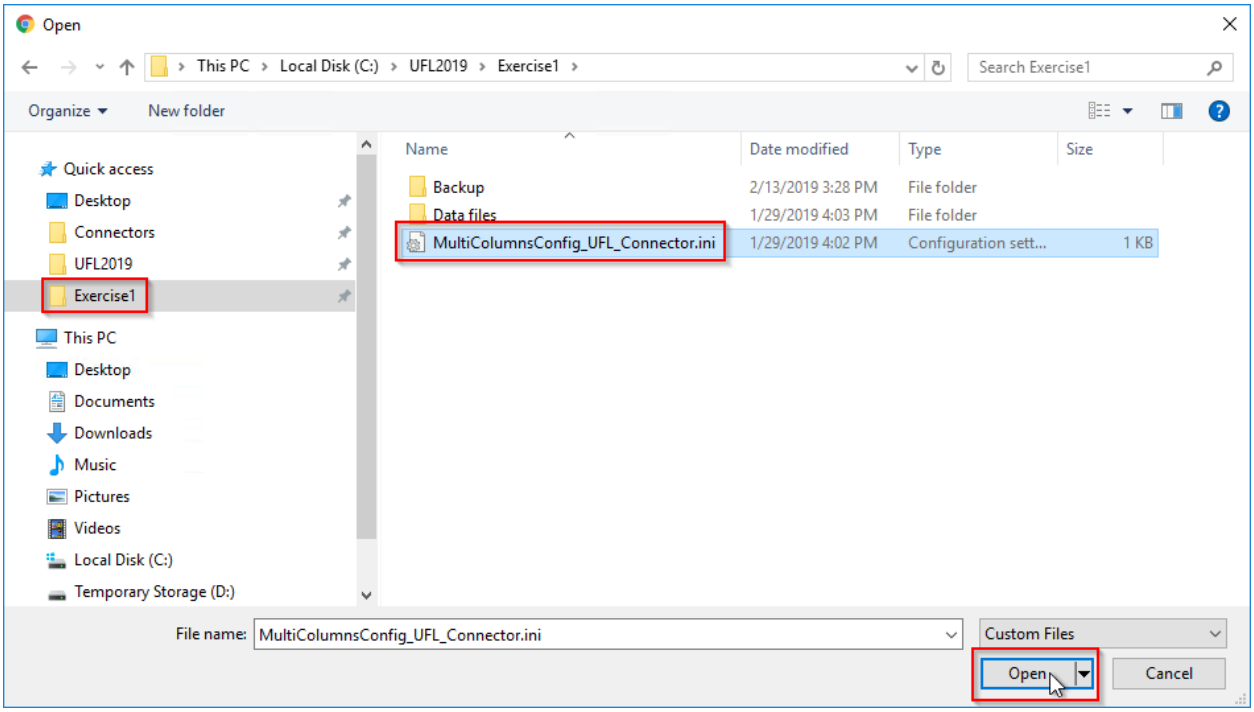


3. Provide any relevant information of choice in the description field.

4. The configuration file for this exercise has already been created. Select **Choose File** and navigate to **Exercise1** from the Quick Access menu. Alternatively, navigate to

   `C:\UFL2019\Exercise1\`

   and select **MultiColumnsConfig_UFL_Connector.ini**. This .ini configuration file contains the parsing logic for the connector and will determine the tag names, values, and (if applicable) which AF elements to create and store events. AF element creation will be discussed in the next exercise.

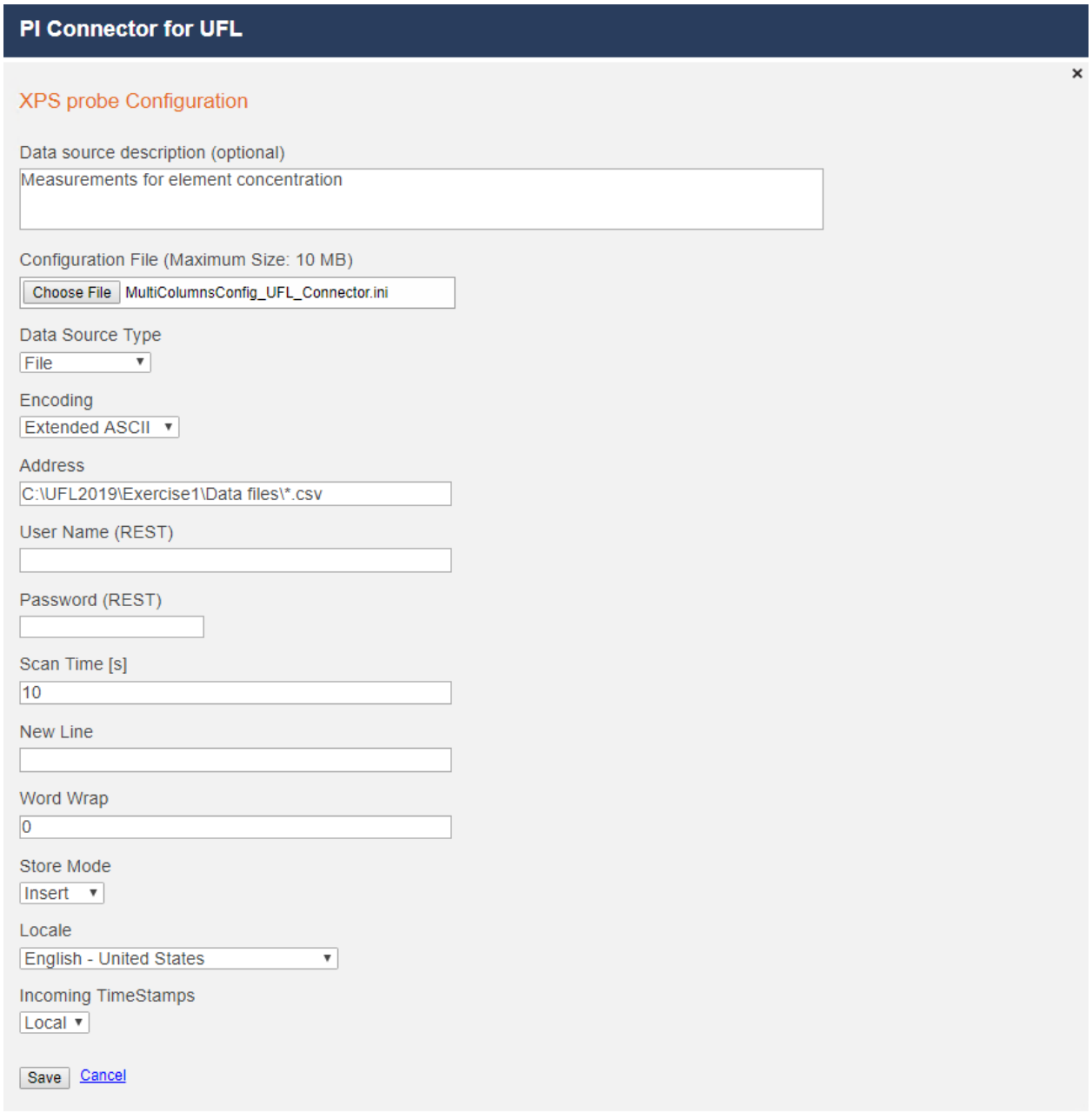


5. The **Data Source Type** can be left as **File**, since the data will be parsed from a file directory that the lab device stores values. The connector can parse files from a local or network file directory.
6. The **Address** field contains the path for input files that will be processed by the connector. Insert the following for the address:

   `C:\UFL2019\Exercise1\Data files\*.csv`

   The asterisk is a wildcard character and any file with the .csv extension will be processed.

7. All other options can be left as the default settings. The final configuration should match the next image.

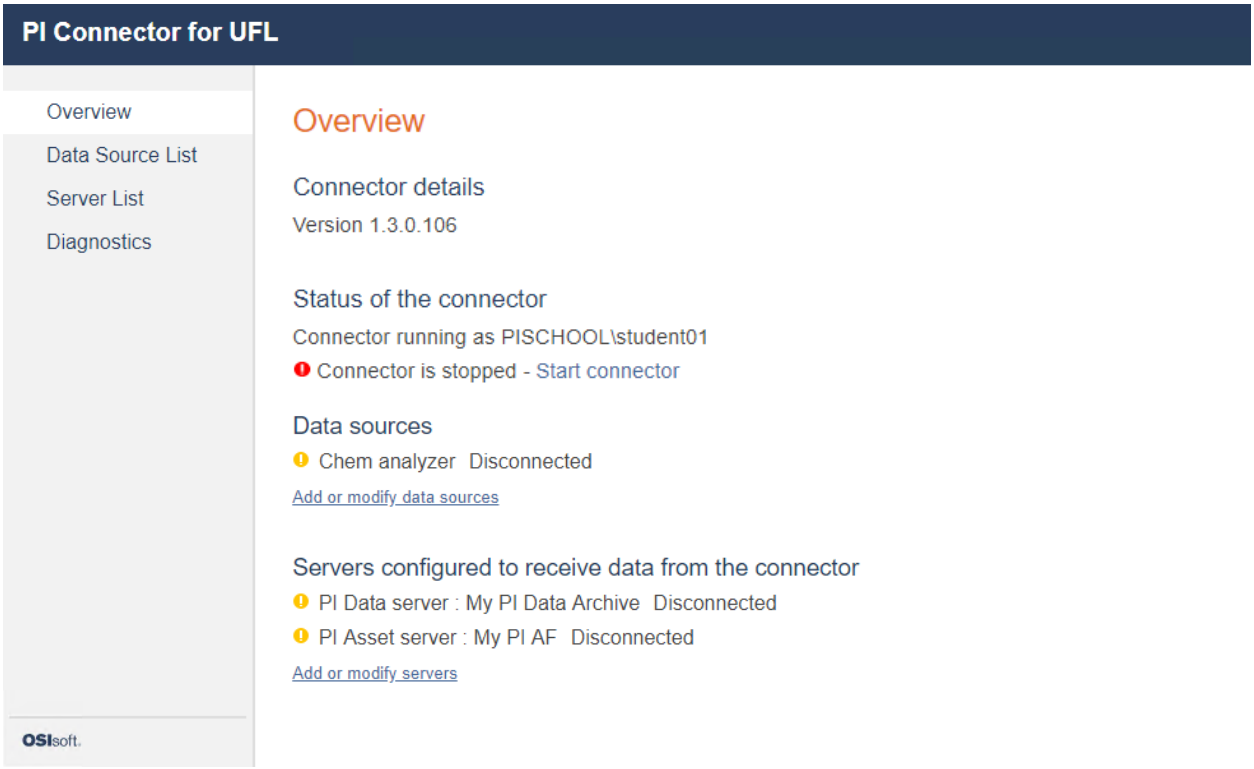


Click **Save** to confirm changes and create the data source.

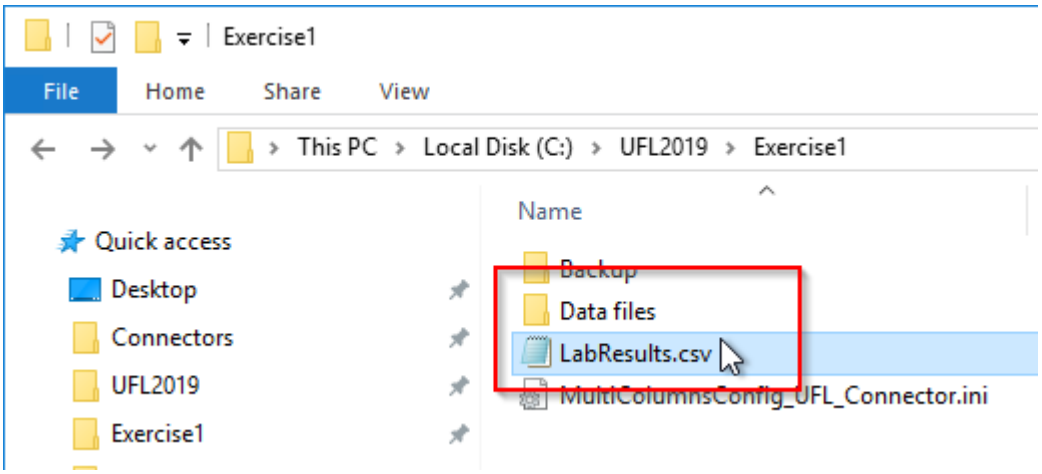
4.4 Step 3: Start the connector

8. Now that the data source is created, click **Start connector** to begin processing incoming or existing data files in the specified directory. The status indicators will turn into green checkmarks.

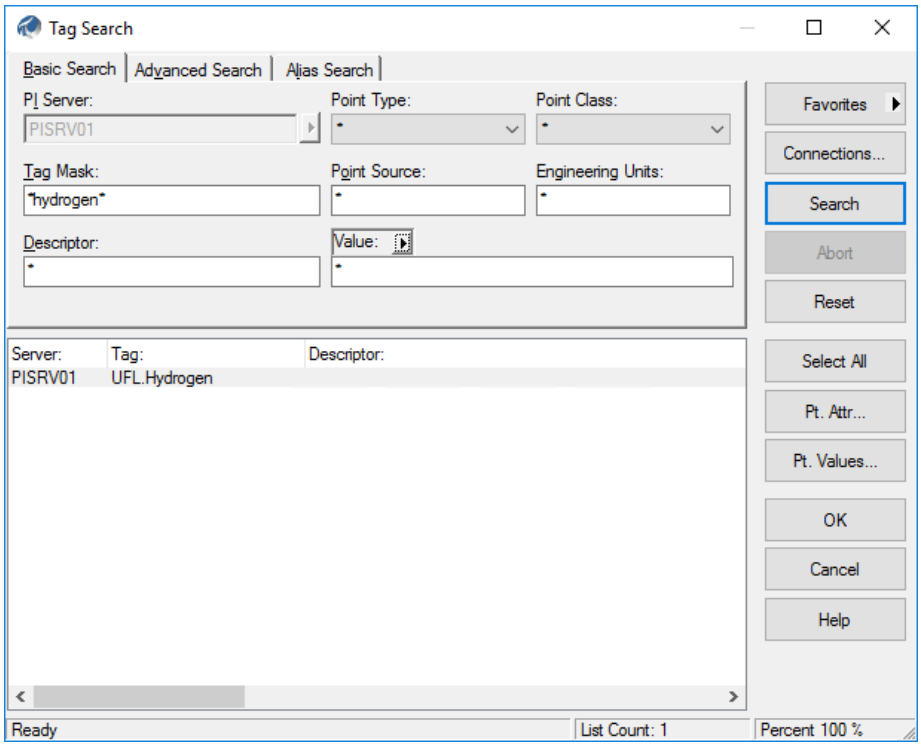
Note:
This does not start or stop the PI UFL Connector Windows service. If the Windows service were not running, the Administration webpage would be unavailable.



9. The connector is now checking the directory at the specified scan time (10 seconds) for new files to process. Move and drag C:\UFL2019\Exercise1**LabResults.csv** to the **Data Files** folder. The **LabResults.csv** file will be removed from the directory as soon as the connector begins processing it.

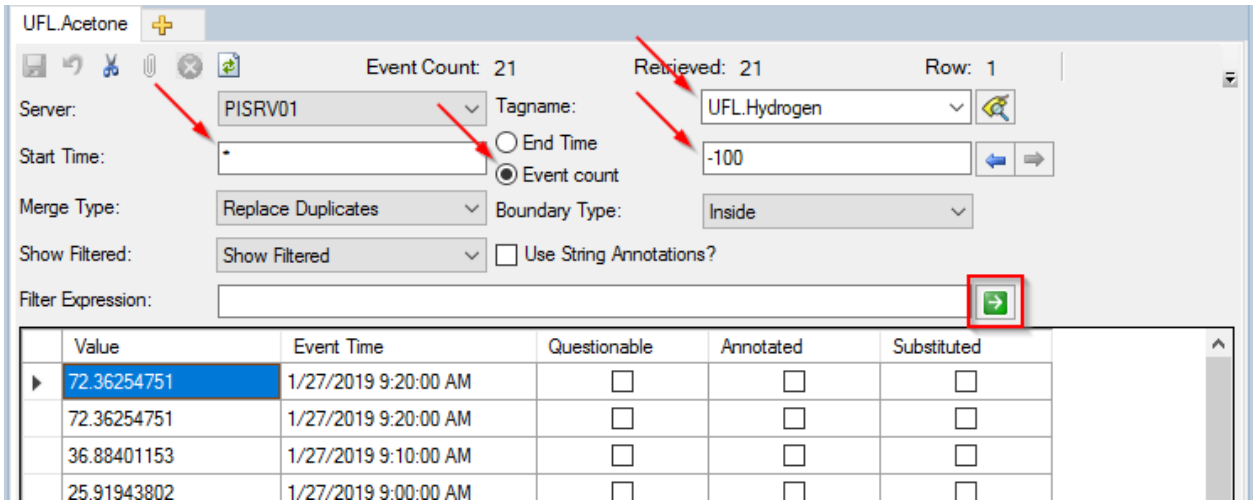


10. To confirm tag creation and data parsing, search for one of the elements in **PI System Management Tools**. Once the application opens, use the left pane to navigate to **Data > Archive Editor**, and select the Tag Search button.



11. Use the tag mask ***hydrogen*** to find the UFL.Hydrogen tag and click **Search**. Double click on the tag to bring the selected item into the Archive Editor.

12. Change the **Start Time** to ***** and the time interval to **Event Count**. This will retrieve the most recent 100 values. Press  to retrieve the values.



13. There are now 20 tags created for these elements. To test the flexibility of the parsing logic, examine the C:\UFL2019\Exercise1**LabResults_100_Elements.csv**. The measurement data contained in this file has 80 more columns\elements. Drop the file in the **Data files** directory and search for tags once the file is processed. There should now be 100 tags for the measurements values.

4.5 INI Configuration file contents

```
[FIELD]
FIELD(1).NAME="TagNames"
FIELD(1).TYPE="Collection"
FIELD(2).NAME="Values"
FIELD(2).TYPE="Collection"
FIELD(3).NAME="Timestamp"
FIELD(3).TYPE="DateTime"
FIELD(3).FORMAT="M/dd/yyyy h:mm"

FIELD(4).NAME="Counter"
FIELD(4).TYPE="Int32"
FIELD(5).NAME="Value"
FIELD(5).TYPE="Number"

[MSG]
MSG(1).NAME="Tags"
MSG(2).NAME="Data"

[Tags]
Tags.FILTER = C1=="T*"
    TagNames = Clear()
    Counter = 0
FOREACH (CsvGetItem(__MESSAGE, ",")) DO
    IF(Counter > 0) THEN
        TagNames = Add(__ITEM)
    ENDIF
    Counter = Counter + 1
ENDFOR

[Data]
Data.FILTER = C1=="*"
    Counter = 0
    Values = Clear()
FOREACH (CsvGetItem(__MESSAGE, ",")) DO
    IF(Counter == 0) THEN
        TimeStamp = __ITEM
    ELSE
        Value = __ITEM
        Values = Add(Value)
    ENDIF
    Counter = Counter + 1
ENDFOR
    StoreEvents(TagNames, ,Timestamp, Values)
```

5. Exercise 2 – Capture data from a REST API endpoint

This exercise showcases the ability of the PI Connector for UFL to make GET requests to an REST API endpoint. The connector will be configured using placeholders to make multiple requests the JSON response will be parsed to create an AF structure that will be visualized with PI Vision. In this exercise, the learning objectives are:

- Understand how to configure the PI Connector to act as a REST client
- Parse JSON formatted objects using native JSON functions
- Create AF Elements, AF Attributes, and AF Element Templates from the configuration file (INI)
- Use the UFL_Placeholder to execute multiple queries within one data source

5.1 Step 1 – Explore the REST API

For the data source in this exercise, OpenWeatherMap's API will be used. The API documentation can be found here:

https://openweathermap.org/api

This exercise focuses on requesting **Current weather data** but the API also makes it possible to capture forecast data (to be stored in future PI Points), historical data, air pollution data and more. For any request made, this REST API requires an API key. An API key is provided to make requests against this API. The API key is located in the following file:

```
C:\UFL2019\Exercise2\RESTAPI\apikey.txt
```

Note:
Anyone can sign up for a free API key that allows up to 60 calls / minute, which should be more than sufficient for weather data.

Start by making a simple request for current weather in San Francisco. The most basic API call uses the following format:

```
api.openweathermap.org/data/2.5/weather?q={cityname}&appid={apikey}&units=imperial
```

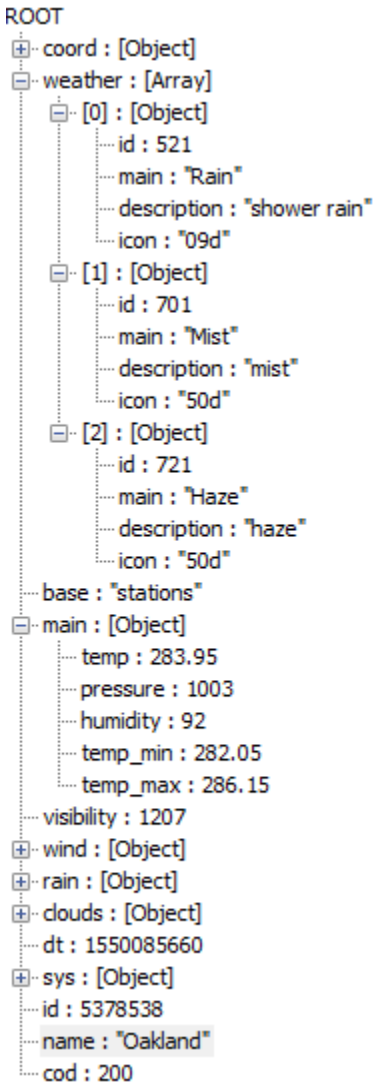
Take a few minutes to execute some queries and refer to the API documentation to see what the string:value pairs stand for.

5.2 Step 2 – Exploring the JSON response

Open Notepad++. Copy the response from the request made in Step 1 and paste it into Notepad++. The response JSON object should look like the following:


```
{"coord":{"lon":-122.27,"lat":37.8},"weather":[{"id":521,"main":"Rain","description":"shower
rain","icon":"09d"},{"id":701,"main":"Mist","description":"mist","icon":"50d"},{"id":721,"main
":"Haze","description":"haze","icon":"50d"}],"base":"stations","main":{"temp":283.95,"pressure
":1003,"humidity":92,"temp_min":282.05,"temp_max":286.15},"visibility":1207,"wind":{"speed":5.
1,"deg":80},"rain":{"1h":0.96},"clouds":{"all":90},"dt":1550085660,"sys":{"type":1,"id":5154,"
message":0.0043,"country":"US","sunrise":1550070016,"sunset":1550108804},"id":5378538,"name":"
Oakland","cod":200}
```


For better readability, the JSON Object can be observed using the JSTool plugin in Notepad++. Navigate to **Plugins** -> JSTool - > JSON Viewer. The JSON Viewer pane opens and allows seeing the content of each object or array:



With the single-city current weather request, the JSON response is an object that also contains multiple child objects. `weather` is an array so the FOREACH() statement would need to be used to parse the array one object at a time. If one were to make a multi-city query, the entire response would be an array (one object per city), so an additional FOREACH() would be required to parse the entire response one city at a time.

5.3 Step 3 – Configure the Data Source

Create a new Data Source with the following parameters:

PARAMETER	VALUE
DATA SOURCE NAME	Weather Monitoring
CONFIGURATION FILE	"C:\UFL2019\Exercise2\INI\UFL_weather.ini"
DATA SOURCE TYPE	REST Client
ENCODING	Extended ASCII
ADDRESS	REST API request built in Step 1
USER NAME / PASSWORD	-
SCAN TIME (S)	20
NEW LINE	-
WORD WRAP	-1

Once the data source has been configured, save it and make sure that your PI Connector is still running. It will start making GET requests on the *Address* defined in the data source configuration and parse the JSON response.

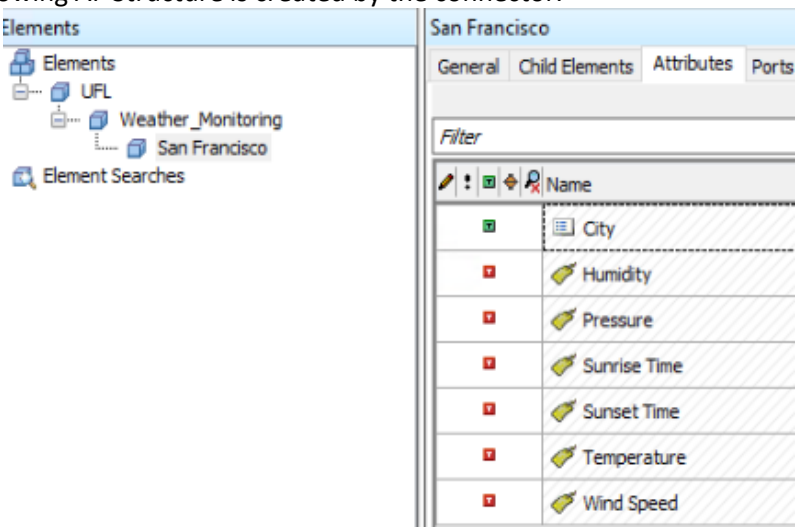
5.4 Step 4 – Inspect the AF Structure created and the PI Vision display

Using **PI System Explorer** (PI System -> PI System Explorer (64-bit)) and navigate to:

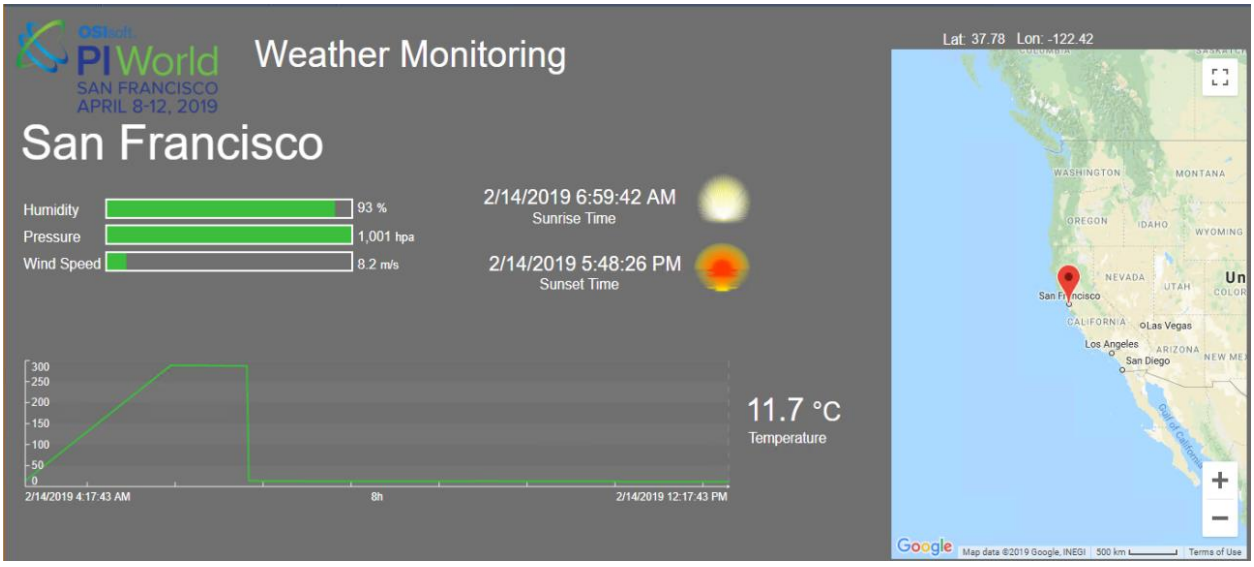
`\\PISRV01\PI UFL Connector\UFL\Weather Monitoring`

Note:
Hit the "Refresh" button at the top to see the database update

The following AF Structure is created by the connector:



As part of this Lab, a PI Vision display was already created. Since the PI Vision display is created based on the template "UFL.WeatherTemplate", it can now be accessed for the city for which the request was made.



5.5 Step 5 – Parsing the JSON response

A sample INI configuration file is provided in "C:\UFL2019\Exercise2\INI\UFL_weather.ini".

Note:
OSIsoft GitHub repo (https://github.com/osisoft/PI-Connector-for-UFL-Samples) has many examples available. It contains complete projects such as the one in this lab, DragonBoard, RaspberryPI, Arduino, and many others.

The INI file uses the native JSON functions described in the **Introduction** section. More details and examples can also be found in the User Guide.

```
[FIELD]
FIELD(1).NAME="City"
FIELD(2).NAME="TagNames"
     TagNames.TYPE="Collection"
FIELD(3).NAME="Values"
     Values.TYPE="Collection"
FIELD(4).NAME="AttributeNames"
     AttributeNames.TYPE="Collection"
FIELD(5).NAME="StaticAttributes"
     StaticAttributes.TYPE="Collection"
FIELD(6).NAME="JSONResponse"
FIELD(7).NAME="temp"
     temp.TYPE="Number"
FIELD(8).NAME="pressure"
     pressure.TYPE="Number"
FIELD(9).NAME="humidity"
     humidity.TYPE="Number"
FIELD(10).NAME="windspeed"
     windspeed.TYPE="Number"
FIELD(11).NAME="sunrise"
     sunrise.TYPE="DateTime"
     sunrise.FORMAT="SECONDS_GMT"
FIELD(12).NAME="sunset"
     sunset.TYPE="DateTime"
     sunset.FORMAT="SECONDS_GMT"
FIELD(13).NAME="GPS_Lon"
     GPS_Lon.TYPE="Number"
FIELD(14).NAME="GPS_Lat"
     GPS_Lat.TYPE="Number"
FIELD(15).NAME="ElementName"

[MSG]
MSG(1).NAME="Data"

[Data]
Data.FILTER=C1=="*"
JSONResponse = __MESSAGE

'Since the city name will be used to define the TagNames, the value is collected
first
City = JsonGetValue(JSONResponse, "name")

'Clear the collections
TagNames = Clear()
Values = Clear()
```

```
AttributeNames = Clear()
StaticAttributes = Clear()

'Build the TagNames collection for the PI Points to be created
Tagnames = Add(city + "_temp")
Tagnames = Add(city + "_pressure")
Tagnames = Add(city + "_humidity")
Tagnames = Add(city + "_windspeed")
Tagnames = Add(city + "_sunset")
Tagnames = Add(city + "_sunrise")

'Build the Static Attributes
GPS_Lat = JsonGetValue(JSONResponse, "coord\lat")
GPS_Lon = JsonGetValue(JSONResponse, "coord\lon")
StaticAttributes = Add("Latitude",GPS_Lat)
StaticAttributes = Add("Longitude",GPS_Lon)
StaticAttributes = Add("City",City)

'Capture the Values for all the variables of interest
'For nested objects, the backslash can be used to get the full path of the
string:value pair
'This array (collection) needs to be in the same order as the tagnames
temp = JsonGetvalue(JSONResponse, "main\temp")
    Values = Add(temp)
pressure = JsonGetValue(JSONResponse, "main\pressure")
    Values = Add(pressure)
humidity = JsonGetValue(JSONResponse, "main\humidity")
    Values = Add(humidity)
windspeed = JsonGetValue(JSONResponse, "wind\speed")
    Values = Add(windspeed)
sunset = JsonGetValue(JSONResponse, "sys\sunset")
    Values = Add(sunset)
sunrise = JsonGetValue(JSONResponse, "sys\sunrise")
    Values = Add(sunrise)

'Build the AttributeNames array
AttributeNames=Add("Temperature")
AttributeNames=Add("Pressure")
AttributeNames=Add("Humidity")
AttributeNames=Add("Wind Speed")
AttributeNames=Add("Sunset Time")
AttributeNames=Add("Sunrise Time")

'Store the values in PI
StoreEvents(TagNames,AttributeNames,,Values)
'Create the Parent Element
StoreElement("Weather_Monitoring")
'Create the AF Element to store the attributes in
ElementName="Weather_Monitoring\" + City
StoreElement(ElementName, "WeatherTemplate",tagnames,StaticAttributes)
```

5.6 Step 6 –Multiple Cities Using the UFL_Placeholder

In the previous steps, the data source was configured for one specific city. To query more than one city, there a couple of possible approaches:

1. If the REST API supports it, make an API call that requests multiple cities at once and returns the result in an array
2. Using UFL_Placeholder, configure the connector to make multiple GET requests on the target REST API to return a series of JSON responses

An example INI, API GET request, and a data file are provided in the Reference list.

In this exercise, users will configure the connector to request a minimum of 5 cities of choice using the second method. Build the query and update the *Address* field in the Data Source Configuration.

Note:
The PI Connector for UFL does not require a restart or to stop the connector to make configuration changes to the data sources.

The following example shows how UFL_Placeholder can be used to turn one REST API request (Address) into multiple queries:

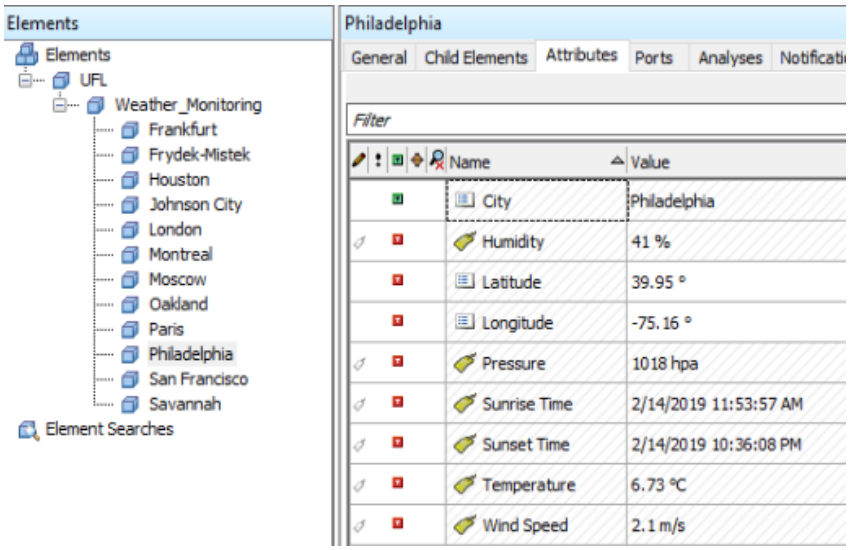
Address =
`api.openweathermap.org/data/2.5/weather?q=UFL_PLACEHOLDER&appid={key}&units=imperial|Montreal|Houston|Prague|Berlin|Tokyo`

This query would execute the following:

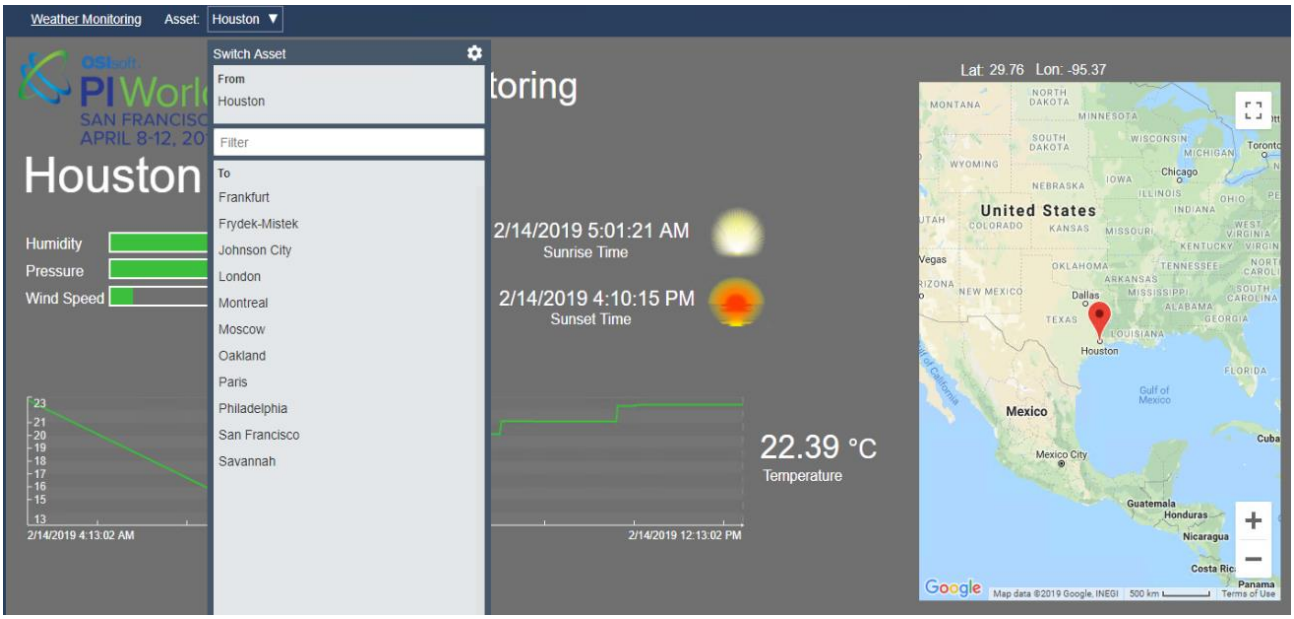
- `api.openweathermap.org/data/2.5/weather?q=Montreal&appid={key}&units=imperial`
- `api.openweathermap.org/data/2.5/weather?q=Houston&appid={key}&units=imperial`
- `api.openweathermap.org/data/2.5/weather?q=Prague&appid={key}&units=imperial`
- `api.openweathermap.org/data/2.5/weather?q=Berlin&appid={key}&units=imperial`
- `api.openweathermap.org/data/2.5/weather?q=Tokyo&appid={key}&units=imperial`

5.7 Step 7 – Explore the Changes Made to the AF Structure and PI Vision Display

With the cities added as placeholders, the connector made multiple GET requests and created the AF hierarchy automatically.



Since the PI Vision display from Step 5 leverages AF templates, all cities are now accessible in the display.



6. Reference list

JSTool

A JavaScript (JSON) tool for Notepad++ (formerly JSMinNpp) and Visual Studio Code.
https://github.com/sunjw/jstoolnpp

Notepad++

Notepad++: a free source code editor which supports several programming languages running under the MS Windows environment.
https://notepad-plus-plus.org/

OpenWeatherMap API

OpenWeatherMap is an online service that provides weather data, including current weather data, forecasts, and historical data via a REST api
https://openweathermap.org/

GitHub repository with examples

Documentation and supporting files to demonstrate usage of the PI Connector for UFL REST endpoint
http://github.com/osisoft/PI-Connector-for-UFL-Samples/

List of publicly available REST APIs

https://www.programmableweb.com/apis/directory


Have an idea how to improve our products?
OSIsoft wants to hear from you!
https://feedback.osisoft.com/

Save the Date!

OSIsoft PI World Users Conference in Gothenburg, Sweden. September 16-19, 2019.

Register your interest now to receive updates and notification early bird registration opening.

https://pages.osisoft.com/UC-EMEA-Q3-19-PIWorldGBG-RegisterYourInterest_RegisterYourInterest-LP.html?_ga=2.20661553.86037572.1539782043-591736536.1533567354