OSIsoft PI World SAN FRANCISCO

MARCH 23-26, 2020

PI World 2020 Lab

Using PI Web API

From Beginner to Advanced

(v1.0)

Published: July 6, 2022

Table of Contents

Contents

1. Introduction

1.1 Overview of Lab

This lab will go through some of the main functionality of the PI Web API, from basic actions like retrieving a set of values, to signing up for streaming updates on a group of attributes. This lab is designed for students who are familiar with the PI System but are new to PI Web API. Advanced programming capabilities are not required, but students should have some familiarity with general programming concepts.

1.2 What is PI Web API?

The PI Web API is a RESTful Web API that allows for programmatic access to the PI System. Through PI Web API, client applications can perform for read/write operations on AF Servers and the PI Data Archives through web requests.

RESTful web services are a common architecture type for modern APIs. In the context of PI Web API, adhering to this type of architecture means that the API is:

- **Stateless** - This means that PI Web API retains no observable knowledge of clients across requests. Each request is an independent transaction between the client and the server.
- **Resource-oriented** - Interaction with the PI Web API is organized around resources. Most important PI and AF objects, such as Asset Servers, Data Servers, points, elements, attributes, event frames, and so on, map to resources in the PI Web API.
- **Navigable by links** - Links capture the organization of the resources exposed by the PI Web API. You're probably familiar with the hierarchical structure of AF objects: Asset Servers contain databases, which contain elements, which contain attributes, and so on. Links returned from requests express these relationships, which make it easy for clients to navigate between related resources.

Comparison to AF SDK

The PI Web API itself uses AF SDK for data access, so its functionality is a subset of what AF SDK provides. This also means that it is not possible for PI Web API to have better performance than AF SDK. However, for many applications the performance difference is small, and PI Web API is much more flexible in terms of what platforms and languages can be used. AF SDK is a .NET library, so it must be run on Windows and in a language that supports loading .NET dlls. On the other hand, PI Web API can be used by any platform and language that supports making HTTP(S) requests.

1.3 Goals for this Lab

The goal of this lab is to expose students to the capabilities of the PI Web API by getting them familiar with navigating the documentation and making requests. After taking this lab, students should understand:

1) How to navigate through various endpoints in a web browser
2) How to make requests through a non-browser HTTP client
3) How to interact with the PI Web API programmatically

1.4 Section outline

- PI Web API Basics (Google Chrome)
  - Become familiar with the documentation
  - Practice retrieving information from different endpoints in the browser

- Constructing a PI Web API request (Postman)
  - Learn the different components of an HTTP request
  - Use functionality that is not possible via the web browser

- Using the AFSearch endpoints (Postman)
  - Understand the different ways that you can search for objects in PI Web API
  - Practice constructing search queries to return objects matching certain criteria

- Web ID 2.0 (Python)
  - Understand the Web ID 2.0 standard
  - Deconstruct WebIds into their component parts
  - Generate WebIds client side

- Batch requests (Python)
  - Look at how the batch endpoint allows for bundling HTTP requests
  - Create parent/child batch requests that have internal references

- Channels (Python)
  - Brief overview of the WebSocket protocol
  - See how the channels feature can be used to monitor for real-time changes

- Stream Updates (Python)
  - Accomplish similar tasks as channels but with purely HTTP requests

1.5 Out of scope topics

There are two important plugins for the PI Web API that we will not be covering in this class: the OMF endpoint and Indexed Search.

OMF Endpoint

The OSIsoft Message Format (OMF) defines a format for data messages that can be read by compliant OSIsoft products. The OMF (OSIsoft Message Format) endpoint was added to PI Web API in the 2019 release and will be the primary method for OMF data ingress to the PI System going forward. When writing large amounts of data through PI Web API it is recommended to use the OMF endpoint, as it was designed specifically for data ingress. If you are interested in learning about OMF and how to develop applications against the PI Web API implementation, please see the Developers Companion Guide: https://explore.osisoft.com/omf/os-isoft-message-for

Indexed Search

PI Indexed Search and PI Indexed Search Crawler are optional features of PI Web API. Indexed search runs as part of PI Web API, while the Search Crawler is an independent service. The Search Crawler service gathers metadata from the PI System and provides items for the Indexed Search for indexing. The pros/cons of searching with Indexed Search are covered in Section 4, but we will focus on other methods of searching and will not discuss configuring the Search Crawler service.

1.6 Virtual Learning Environment

The Virtual Learning Environment for this lab is a very simple setup with 3 machines.

PISCHOOL Domain:

PICLIENT01 – This is our client machine that will be where we complete each of the exercises for the lab. The three client applications that we will use are already installed: Google Chrome, Postman, and Visual Studio Code (Python).

PISRV01 – This is an all-in-one PI Server machine that we will be connecting to and retrieving data from. It hosts the PI Web API, PI Data Archive, and PI Asset Framework Server.

PIDC – Domain Controller (no interaction necessary for this lab)

The user account that we will be using is **PISCHOOL\student01**. The password for this account will be given by the lab instructor.

Exercise 1 – Exploring the PI Web API

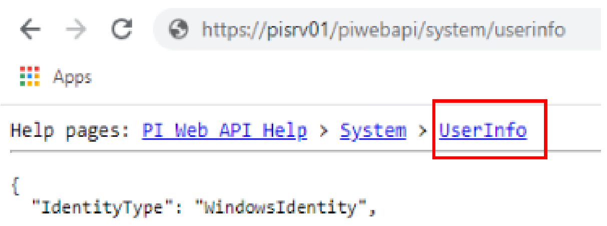
1) Log into VLE
    a. Link to environment sent in email

2) Open Google Chrome

3) Open the home page of PI Web API
    a. https://pisrv01.pischool.int/piwebapi

4) Find help documentation
    a. https://pisrv01.pischool.int/piwebapi/help
    b. https://techsupport.osisoft.com/Documentation/PI-Web-API/help/

5) Which Windows account is being used?
    a. Go to home page
    b. Click System
    c. Click the UserInfo link
        - ➢Look at the URL
        - ➢Click on the link to the help page for this endpoint



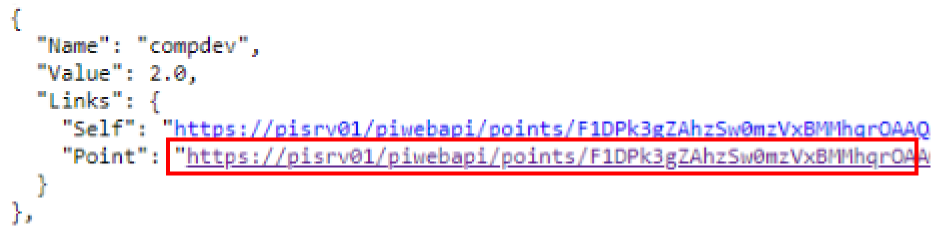
```
Help pages: PI Web API Help > System > UserInfo

{
  "IdentityType": "WindowsIdentity",
```

6) How long has the PI Web API been running?
    a. Go back to System default endpoint
    b. Click the Status link

7) Find information about the PI tag named **sinusoid**
    a. Go to the home page
    b. Click the DataServers link
    c. Click on Points link for the DataServer **PISRV01**
    d. Press Ctrl+F and search for sinusoid
        - ➢What is the Descriptor for this tag?
    e. Click on Self link

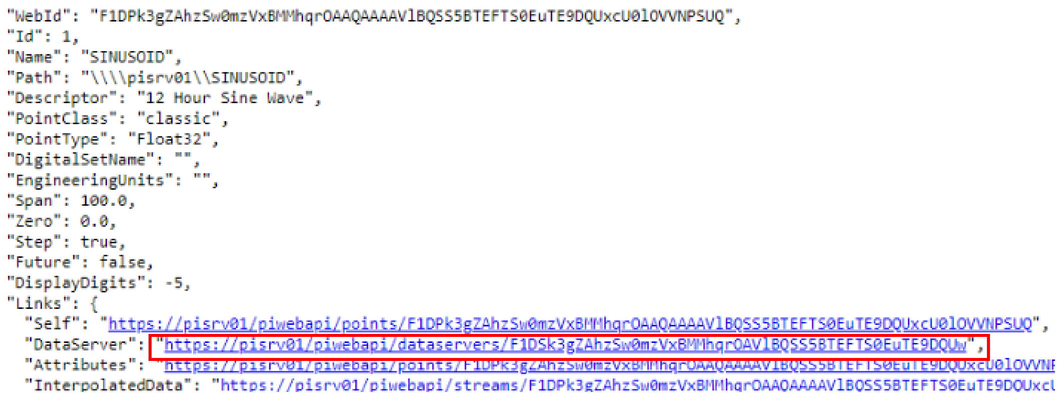
f. Click on the Attributes link
   - ➢What is the Compression Deviation?

g. Use links to navigate back to the PISRV01 DataServer
   - ➢Click on Point link



```
{
  "Name": "compdev",
  "Value": 2.0,
  "Links": {
    "Self": "https://pisrv01/piwebapi/points/F1DPk3gZAhzSw0mzVxBMMhqrOAAQ
    "Point": "https://pisrv01/piwebapi/points/F1DPk3gZAhzSw0mzVxBMMhqrOA
  }
},
```

   - ➢Click on DataServer link



```
"WebId": "F1DPk3gZAhzSw0mzVxBMMhqrOAAQAAAAV1BQSS5BTEFTS0EuTE9DQUxcU0lOVVNPSUQ",
"Id": 1,
"Name": "SINUSOID",
"Path": "\\\\pisrv01\\SINUSOID",
"Descriptor": "12 Hour Sine Wave",
"PointClass": "classic",
"PointType": "Float32",
"DigitalSetName": "",
"EngineeringUnits": "",
"Span": 100.0,
"Zero": 0.0,
"Step": true,
"Future": false,
"DisplayDigits": -5,
"Links": {
  "Self": "https://pisrv01/piwebapi/points/F1DPk3gZAhzSw0mzVxBMMhqrOAAQAAAAV1BQSS5BTEFTS0EuTE9DQUxcU0lOVVNPSUQ",
  "DataServer": "https://pisrv01/piwebapi/dataservers/F1DSk3gZAhzSw0mzVxBMMhqrOAV1BQSS5BTEFTS0EuTE9DQUw",
  "Attributes": "https://pisrv01/piwebapi/points/F1DPk3gZAhzSw0mzVxBMMhqrOAAQAAAAV1BQSS5BTEFTS0EuTE9DQUxcU0lOVVNP
  "InterpolatedData": "https://pisrv01/piwebapi/streams/F1DPk3gZAhzSw0mzVxBMMhqrOAAQAAAAV1BQSS5BTEFTS0EuTE9DQUxcl
```

8) Navigate AF hierarchy
   a. Go to the home page
   b. Click on the AssetServers link
   c. Click on Databases link for **PISRV01**
      - ➢How many AF Databases do we have?
   d. Click on the Elements Link for the **NuGreen** database
      - ➢How many root elements are there in this Database?
   e. By clicking the Elements links, navigate to **\\PISRV01 NuGreen\NuGreen\Houston\Cracking Process\Equipment\B-210**
   f. Click on the Value link
      - ➢What is the current timestamp and value for the Fuel Savings attribute?

9) PI Web API configuration element
   a. Open PI System Explorer
   b. Go to Configuration database
   c. Find the element **\\PISRV01\Configuration\PI Web API\PISRV01\System Configuration**
   d. What authentication methods are enabled for this PI Web API instance?

2. PI Web API Basics

In this section we will examine the core concepts of PI Web API and see how we can change what information is returned.

2.1 Controllers

Controllers are the top-level groupings that provide access to different types of objects or functionality within the PI Web API. Some of the most commonly used controllers are listed below:

- Attribute (AF Attributes)
- AssetServer (AF Servers)
- AssetDatabase (AF Databases)
- Batch (used for bulk operations – covered in section 6)
- DataServer (PI Data Archives)
- Element (AF Elements)
- EventFrame (AF Event Frames)
- Point (PI Data Archive Tags)
- Stream
- StreamSet

The Stream and StreamSet controllers are particularly important. They provide the functionality for reading/writing time-series data.

A stream is defined as either a tag on the PI Data Archive or an AF Attribute that has a Data Reference configured (i.e. static AF Attributes are not a stream). A stream set is a collection of multiple streams. The streams in a stream set may be defined by a common parent element/attribute or can be a grouping of unrelated streams.

2.2 Methods

Each controller has one or more methods that are associated with it. The method describes the actual action we want to take. Some examples of methods for the Element controller are listed below:

- Get (retrieve information about a single AF Element)
- GetByPath (retrieve information about an AF Element by supplying its path)

- Update (make a change to the AF Element definition)
- CreateAttribute (create a new AF Attribute on the specified element)
- GetElementsQuery (return information for elements that match a search criteria)

2.3 URL parameters

URL parameters are used to pass additional values to a method to control what is returned. Each method accepts a specific group of parameters, some of which may be required and others may be optional (have a default value).

Some important URL parameters described below:

- WebId (unique identifier for primary resources – covered in section 5)
- SelectedFields (tells the PI Web API to omit unwanted fields from the response body)
- MaxCount (controls the maximum number of results to be returned – default is 1000)
- TimeZone (which time zone the PI Web API will interpret the supplied timestamps)
- BufferOption (controls whether or not to use PI Buffer Subsystem when writing data)

Exercise 2 – Using URL parameters

Complete the following tasks using Google Chrome.

1) Use the Element controller's GetByPath endpoint to retrieve the AF element **\\PISRV01\NuGreen\NuGreen\Tucson\Distilling Process\Equipment\F-110**

2) Get the same element by passing the WebId
    a. You can do this manually by using the Element controller's Get method or by clicking the Self link from the previous question

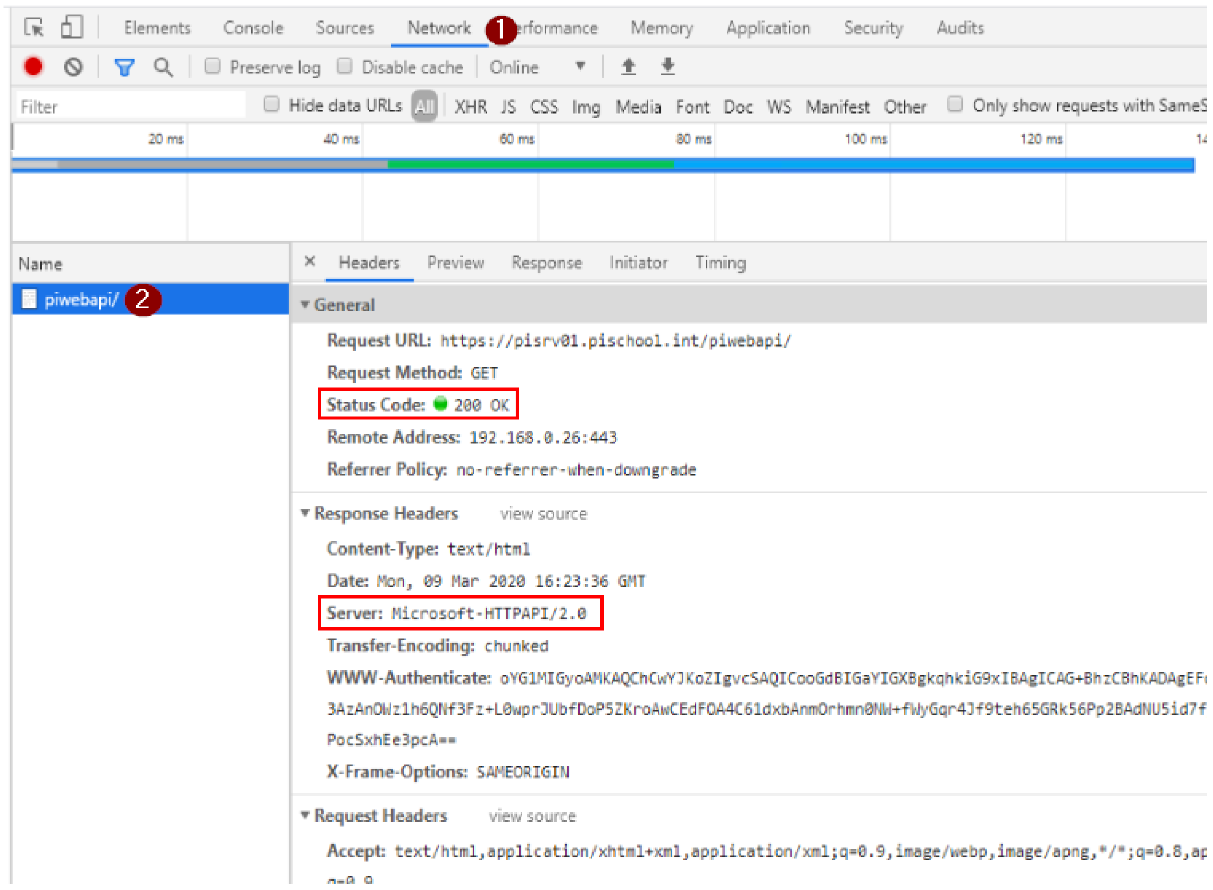
3) Get recorded values for **Motor Amps** attribute
    a. Use the Stream controller's GetRecorded method
    b. Change the time range the from 2 days ago to 1 day ago using the startTime and endTime URL parameters
        - ?startTime=*-2d&endTime=*-1d
    c. Get up to 5 values using the maxCount URL parameters
        - &maxCount=5
    d. Return the same query, but only include values above 75 using the filterExpression URL parameter
        - &filterExpression='.'>75

4) Get interpolated values for this attribute over the last day at 1-hour intervals for the same attribute
    a. Use the Stream controller's GetInterpolated method
    b. The following URL parameters can be used to specify the request details
        - ?startTime=*-1d&endTime=*&interval=1h
    c. Change the query so that only the timestamps and values are returned by using the selectedFields URL parameter
        - &selectedFields=Items.Timestamp;Items.Value;

5) Get the average value for the same attribute over the last week
    a. Use the Stream controller's GetSummary method
    b. The summaryType URL parameter defines the type of summary to calculate
        - ?summaryType=Average

6) Open Chrome DevTools
    a. Press F12
    b. Refresh the page or go to any PI Web API endpoint of your choosing
    c. What status code was returned in the response?

- ➢Go to Network tab
- ➢What is the value of the Server header that is returned in the Response?



Elements Console Sources Network 1 Performance Memory Application Security Audits
Preserve log Disable cache Online
Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other Only show requests with SameS
20 ms 40 ms 60 ms 80 ms 100 ms 120 ms
Name
piwebapi/ 2
Headers Preview Response Initiator Timing

▾ General

Request URL: https://pisrv01.pischool.int/piwebapi/
Request Method: GET
Status Code: 200 OK
Remote Address: 192.168.0.26:443
Referrer Policy: no-referrer-when-downgrade

▾ Response Headers view source

Content-Type: text/html
Date: Mon, 09 Mar 2020 16:23:36 GMT
Server: Microsoft-HTTPAPI/2.0
Transfer-Encoding: chunked
WWW-Authenticate: oYG1MIGyoAMKAQChCwYJKoZIgvcSAQICooGdBIGaYIGXBgkqhkiG9xIBAgICAG+BhzCBhKADAgEF
3AzAnOWz1h6QNf3Fz+L0wprJUbfDoP5ZKroAwCEdFOA4C61dxbAnmOrhmn0NW+fWyGqr4Jf9teh65GRk56Pp2BAdNU5id7f
PocSxhEe3pcA==
X-Frame-Options: SAMEORIGIN

▾ Request Headers view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,ap
q=0.9

3. Constructing PI Web API Requests

In this section we will examine the building blocks of an HTTP request and see how to construct requests that make changes to the PI System.

3.1 HTTP verbs

Through the browser, the only type of HTTP request we can make is a GET request. However, to make any changes to the PI System (not purely retrieving data) we must use other HTTP Verbs. Each method in the PI Web API requires a specific verb, depending on the action that is being taken:

- GET – read actions
- POST – create actions
- PUT – update actions (with complete definition of the resource)
- PATCH – update actions (with partial definition of the resource)
- DELETE – delete actions

3.2 Headers

Headers are key/value pairs that accompany the request and contain metadata. Some headers are usually supplied automatically by the client/library. For example, when we make requests through a web browser, headers like Authorization generated automatically for us. However, some headers may need to be specified explicitly depending on the request and the desired behavior. Some important headers are described below:

- Authorization

This contains the authentication scheme (Basic, Negotiate, Bearer) as well as the corresponding credentials for verifying the identity of the requester.

- Cache-Control

The PI Web API uses an AFCache to temporarily store AF SDK objects that have already been retrieved. By default, the cache will refresh every 5 minutes. This header allows the requester to specify a maximum age (in seconds) for cached objects. This ensures that anything returned by the PI Web API was retrieved from the server within the specified time. This header does not affect timeseries data, which is never cached and is always retrieved at the request time.

✓ Best Practice	Excessive use of the *cache-control: no-cache* header can have detrimental effects on performance, especially for repeated requests. This header should be used cautiously and only when necessary.

- X-Requested-With

This header must be included with any POST, PUT, PATCH, or DELETE request as part of CSRF (Cross Site Request Forgery) defense. The actual value of the header does not matter, just that it is included.

- Content-Type

This is a standard http header the describes the format of the request body (when supplied). For PI Web API we will set this to *application/json*.

3.3 Request body

Unlike GET requests, which contain all of the request information in the URL and headers, other request types require a payload to contain the data that is to be created or changed. This payload is contained in the body of the HTTP request. PI Web API accepts and returns content in the JSON (JavaScript Object Notation) form. JSON is a commonly used format of representing objects in a human readable text format. Below, here is an example of JSON data you may see returned from PI Web API.

```
{
  "Links": {},
  "Items": [
    {
      "Timestamp": "2020-01-15T18:05:11Z",
      "Value": 7.873338,
      "UnitsAbbreviation": "",
      "Good": true,
      "Questionable": false,
      "Substituted": false,
      "Annotated": false
    },
    {
      "Timestamp": "2020-01-15T18:13:41Z",
      "Value": 9.985083,
      "UnitsAbbreviation": "",
      "Good": true,
      "Questionable": false,
      "Substituted": false,
      "Annotated": false
    }
  ],
  "UnitsAbbreviation": ""
}
```

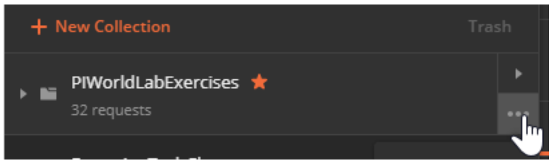
JSON is made up of name/value pairs. For example, “Good” is a name with value “true”. Names are strings, but the values can have different types, such as numbers, strings and true/false. Also important is that each name is unique within an object (delimited by curly braces). The other data type we see in this example is an ordered array (delimited by square braces), for example the value corresponding to the name “Items” is an array containing two objects, each representing a data event.

Exercise 3 – Make requests in Postman

This exercise and Exercise 4 will be using an application called Postman, which is another tool for making HTTP requests. Unlike a web browser, we can customize our requests to have actions besides GET, attach headers, and include body content. A quick introduction to Postman can be found in the Appendix of this workbook.

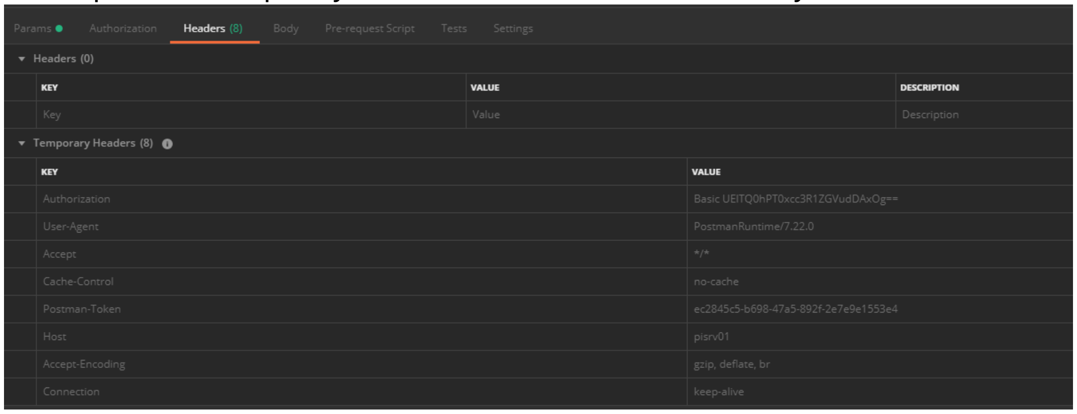
Open Postman and look at the collection called "**PIWorldLabExercises**". You will modify each of the requests to retrieve the data specified in each question. Anywhere that you see *TODO* is something that must be completed by you.

Postman does not have the built-in ability to do Kerberos (Negotiate) authentication, so for these exercises we will use Basic authentication instead. To globally set the Authentication header click "…" on the right side of the top-level folder.



Navigate to the Authorization tab and choose to add a Basic Auth header to all requests. Specify the username **PISCHOOL\student01** and enter the password that you were given. This will set the Authorization header for all requests in this collection.

1) Get the **NuGreen** AF database
    a. Use the AssetDatabase controller's GetByPath method
    b. Expand the temporary headers to see what was added by Postman automatically



Headers (0)

KEY	VALUE	DESCRIPTION
Key	Value	Description

Temporary Headers (8)

KEY	VALUE
Authorization	Basic UEITQ0hPT0xcc3R1ZGVudDAxOg==
User-Agent	PostmanRuntime/7.22.0
Accept	*/*
Cache-Control	no-cache
Postman-Token	ec2845c5-b698-47a5-892f-2e7e9e1553e4
Host	pisrv01
Accept-Encoding	gzip, deflate, br
Connection	keep-alive

2) Create a new root AF element in that database called **OldGreen** based on the element template "Enterprise"
    a. Use the AssetDatabase controller's CreateElement method

3) Create new child element under OldGreen named **Generator**
    a. Use the Element controller's CreateElement method

4) Create a new static attribute on the Generator element named **Power**
    a. Use the Element controller's CreateAttribute method
    b. Attribute Type should be "Int32", and the units should be in kilowatt

5) Create a new PI tag called **PIWorldTag**
    a. Use the DataServer controller's CreatePoint method
    b. The PointClass should be "classic", and the PointType should be "Int32"

6) Turn compression off for this newly created tag
    a. Use the Point controller's UpdateAttributeValue method
    b. The name parameter for this request will be **compressing**
    c. The value to pass in the body of the request to turn off compression is **0**
    d. Unlike most other endpoints, for this method the body does not need to be in JSON format

7) Update the Power attribute to point to the newly created PI tag
    a. Use the Attribute controller's Update method
    b. Change the DataReferencePlugin to "PI Point"
    c. Change the ConfigString to "**\\\\PISRV01\\PIWorldTag**"

8) Write a set of 3 different values to this new tag using Tag's WebId
    a. Use the Stream controller's UpdateValues method

9) Query for those values using AF Attribute's WebId
    a. Use the Stream controller's GetRecorded method

10) Demonstrate caching behavior
    a. Use the Element controller's GetElements method to retrieve child elements of **\\PISRV01\NuGreen\OldGreen** in Postman using the WebId
    b. See that the element **Generator** has no description
    c. Open PI System Explorer, and a description for the Generator element
    d. Re-run the Get request and see that the description change is not reflected
    e. Now add a **cache-control: no-cache** header and see that the change to the description that was made in PI System Explorer is now reflected

4. Searching with PI Web API

A common task that you may need to perform when programming against the PI System is to search for objects that match specific criteria. The PI Web API provides 3 ways to execute search actions within the PI Web API:

1) Traditional endpoints with filters
2) Using the Search controller
3) Using Query methods that utilize the AFSearch classes in AF SDK

4.1 Traditional endpoints with filters

Many core controllers support various types of filtering through URL parameters. For example, the AssetDatabase controller's GetElements method allows you to search for elements that match filter criteria in an AF database.

Example: To search for elements in a database based on a template called "PumpTempate" and have the letter "b" in their name, you could do the following:

GET {baseURL}/assetdatabases/{webId}/elements?nameFilter=*b*&templateFilter=PumpTemplate&searchFullHerarchy=true

4.2 Using the Search controller

This entire controller is dedicated to searching. It utilizes an indexed search and requires databases to be crawled by the PI Web API Search Crawler before performing queries.

Example To search for elements in a particular database based on a template called "PumpTempate" and have the letter "b in their name, you could do the following:

GET {baseURL}/search/query?q=name:*b* AND afelementtemplate=PumpTemplate AND scope=MyAFDatabase

4.3 Using the AFSearch methods

Starting with PI Web API 2017 R2, several new searching methods were introduced which make use of the AFSearch namespace in AF SDK. These methods use the AFSearch query syntax and provide a great deal of flexibility in building the search criteria.

As of PI Web API 2019, there are AFSearch methods for the following objects:

- Analyses (GetAnalysesQuery)
- Attributes (GetAttributesQuery)
- Elements (GetElementsQuery)
- Event Frames (GetEventFramesQuery)
- Notification Rules (GetNotificationRulesQuery)
- Analysis Templates (GetAnalysisTemplateQuery)
- Notification Contact Template (GetNotificationContactTemplatesQuery)
- Notification Rule Templates (GetNotificationRuleTemplatesQuery)

Example: To search for elements in a particular database based on a template called "PumpTempate" and have the letter "b in their name, you could do the following:

GET {baseURL}/elements/search?databaseWebId={webId}&query=Name:=*b* AND Template:=PumpTemplate

4.4 Benefits of the AFSearch methods

The AFSearch methods provide significant advantages over the other two search types.

Compared to using filters on the traditional methods, the AFSearch Query endpoints have many performance improvements. Using the traditional methods with URL filters often means that result filtering must be done within the PI Web API itself rather than the resource server. The AFSearch methods are also able to make use of better caching both within PI Web API and on the AF Server.

Searching via the Search controller generally fast but requires much more setup and configuration. Each AF database or PI Data Archive that is going to be searched must be configured to be crawled by the PI Web API Search Crawler, which creates index files for each server. This introduces more potential problems and administrative burden. For many applications the AFSearch endpoints are similarly performant and the more robust option.

Exercise 4 – Making queries with the AFSearch methods

In Postman, open the collection subfolder for Exercise 4 **–** AFSearch endpoints.

1) Using the Element controller's GetElementsQuery method, search for AF elements that meet the following
    a. In the **NuGreen** database
    b. Name contains the characters "**b-2**"
    c. Element template is "**Boiler**"

2) Again, using the Element controller's GetElementsQuery method, search for AF elements that meet the following
    a. In the **NuGreen** database
    b. Element template is "**Pump**"
    c. The pump's Process is "**'Extruding Plant'**"
    d. The pump's Manufacturer is "**Sterns**"

3) Using the Attribute controller's GetAttributesQuery method, search for AF attributes that meet the following
    a. In the **NuGreen** database
    b. Associated element is based on the template "**Unit**"
    c. Name contains the characters "**flow**"

4) Again, using the Attribute controller's GetAttributesQuery method, search for AF attributes that meet the following
    a. In the **NuGreen** database
    b. Associated element is based on the template "**Boiler**"
    c. Under the root element **NuGreen\Tucson**

5) Using the EventFrame controller's GetEventFramesQuery method, search for EventFrames that meet the following
    a. In the **NuGreen** database
    b. Event Frame is based on the template "**OSIUnitProcedure**"
    c. BatchID attribute is "**1004"**

6) Again, using the EventFrame controller's GetEventFramesQuery method, search for the **5** most recent Event Frames that meet the following
    a. In the **NuGreen** database
    b. Severity is **Critical**

5. WebId 2.0

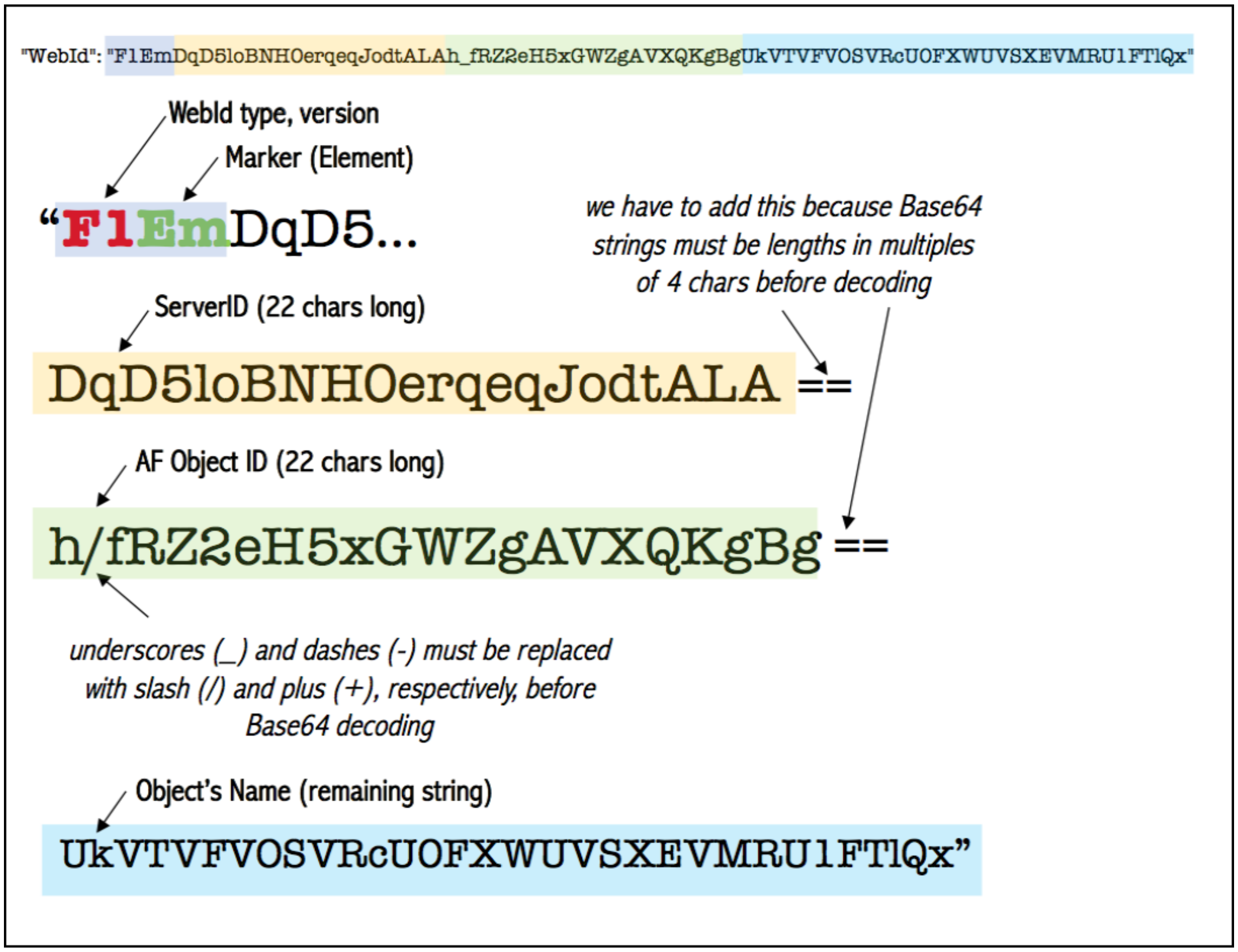
As we have seen previously, every primary resource within the PI Web API has a unique identifier called a WebId. These WebIds are persistent, and therefore can be stored by a client for future use. PI Web API Version 2017 R2 introduced a new version of WebId (2.0), which adheres to an open standard and allows the possibility for WebIds to be constructed by a client. Prior to this version 1.0 WebIds were opaque and had no publicly exposed standard for understanding their structure.

5.1 WebId types

There are 5 different types of WebIds that are available. The type used will affect what types of changes the WebId is resilient to. For example, if the name of an AF element changes, a previously cached PathOnly WebId will no longer work, but an Full or IDOnly WebId would still return the element. In many methods, you can specify the type of WebId that you want to be returned to us by using a URL parameter (e.g. ?webIdType=PathOnly).

Full	Encodes all information, similar to WebID version 1.0. Full WebIDs are longer, but are more resistant to items being moved, renamed, or deleted.
IDOnly	Encodes only object IDs into the WebID. IDOnly WebIDs are shorter, and will always refer to the same item, even if it is moved. However, IDOnly WebIDs will no longer be valid if the item is deleted.
PathOnly	Encodes only path information into the WebID. PathOnly WebIDs will always refer to the same location in the AF hierarchy, regardless of which item is located there.
LocalIDOnly	This type is similar to IDOnly, but is dedicated to resources on the local (relative to the PI Web API instance) asset or data server. The local asset/data server ID and path information is left out. Note: This type is not persistent or unique, and could represent different items on different servers. **It is not compatible with load balancer configurations.**
DefaultIDOnly	This type is similar to IDOnly, but is dedicated to resources on the default asset server or data server. The default asset/data server ID and path information is left out. Note: This type is not persistent or unique, and could represent different items on different servers. **It is not compatible with load balancer configurations.**

The diagram below shows how a Full WebId for an AF element can be broken down into its component parts:


"WebId": "F1EmDqD5loBNHOerqeqJodtALAh_fRZ2eH5xGWZgAVXQKgBgUkVTVFVOSVRcUOFXWUVSXEVMRU1FTlQx"
WebId type, version
Marker (Element)
"F1EmDqD5...
we have to add this because Base64 strings must be lengths in multiples of 4 chars before decoding
ServerID (22 chars long)
DqD5loBNHOerqeqJodtALA ==
AF Object ID (22 chars long)
h/fRZ2eH5xGWZgAVXQKgBg ==
underscores (_) and dashes (-) must be replaced with slash (/) and plus (+), respectively, before Base64 decoding
Object's Name (remaining string)
UkVTVFVOSVRcUOFXWUVSXEVMRU1FTlQx"

The full specification for WebId 2.0 can be found on PI Square: https://pisquare.osisoft.com/community/developers-club/blog/2018/01/26/pi-web-api-web-id-20-specification-tables.

Exercise 5a – Decode WebIds

1. On PICLIENT01, open the file **Exercise 5a – Decode WebIds.py** in Visual Studio Code. This file contains the outline of some of the steps needed to decode a WebId (Full, Path Only, or ID Only).

   This script makes use of another package called webidhelper. This package has some static functions that perform lookups and decode different portions of the WebId.

2. Edit the script to make a request that returns the element **\\NuGreen\NuGreen\Wichita** and decode the full WebId.

3. Edit the URL from your previous request to have it return a PathOnly WebId and decode the components.

4. Edit the URL again to return the IDOnly form of the WebId and decode the components.

BONUS: Currently, this script only works for some object types like AF Elements or Event Frames. Edit the script so that it can decode Full AF Attribute WebIds as well.

Exercise 5b – Generate a WebId

1. On PICLIENT01, open the file **Exercise 5b – Generate a WebId.py** in Visual Studio Code.

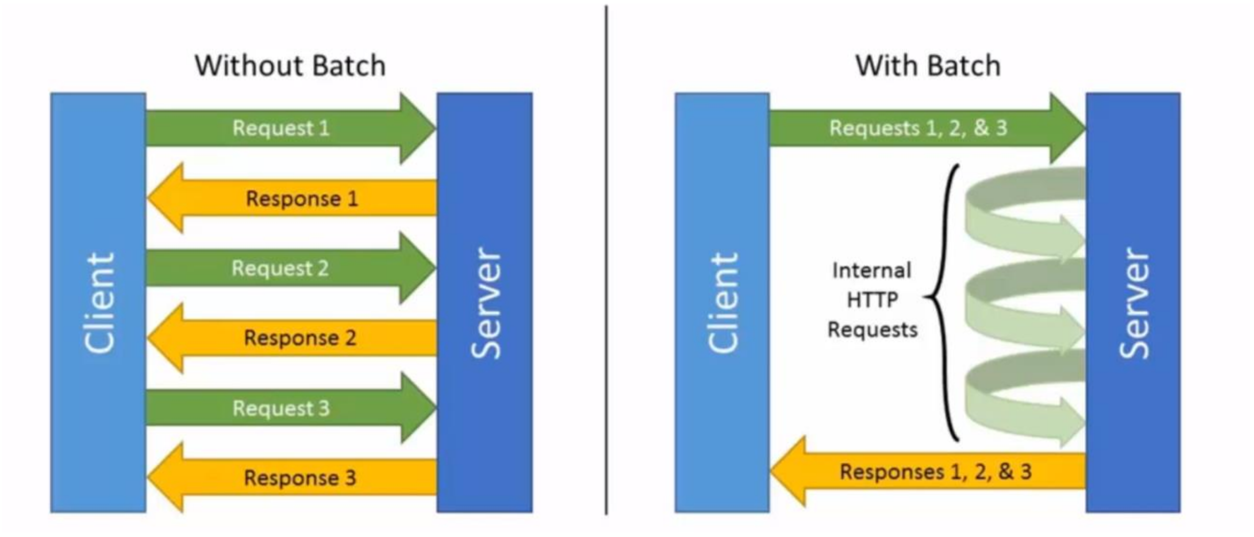
2. Complete the script in to generate a PathOnly WebId for the element **\\PISRV01\NuGreen\Tuscon\Distilling Process\Equipment\P-871** and use it to make a request to the Element controller.

3. Edit the script again to generate a PathOnly WebId for the PI tag **CDT158** and use it to make a request to the Stream controller for recorded values.

BONUS: Edit the script again to generate a WebId for the AF attribute **\\PISRV01\NuGreen\Tuscon\Distilling Process\Equipment\P-871|Process Feedrate** and use it to make a request to the Stream controller for the current value.

6. Batch requests

Every HTTP request has some overhead associated with it, both in terms of processing and network traffic. In order to reduce repetitive processing and effects of latency, we can use the Batch controller to bundle multiple request together and send them to PI Web API in one action. This can result in substantial performance improvements.



6.1 Structure of a batch request

The Batch controller only has one method: Execute. This endpoint accepts POST requests, and the individual actions to be taken are specified within the request body

The JSON body of the request is a list of objects that each correspond to one individual sub-request. Within the sub-request object, individual name/value pairs specify request details (Method, Resource, Content, Headers, etc).

An example request body is below:

```
{
  "SubRequestID1": {
    "Method": "GET",
    "Resource": "https://servername/piwebapi "
  },
  "SubRequestID2": {
    "Method": "GET",
    "Resource": "https://servername/piwebapi/assetservers?
selectedFields=Items.Name;Items.WebId"
  }
}
```

The response body, would then look like:

```
{
    "SubRequestID1": {
        "Status": 200,
        "Headers": {
            "Content-Type": "application/json; charset=utf-8"
        },
        "Content": {
            "Links": {
                "Self": "https://servername/piwebapi/",
                "AssetServers": "https://servername/piwebapi/assetservers",
                "DataServers": "https://servername/piwebapi/dataservers",
                "Omf": "https://servername/piwebapi/omf",
                "Search": "https://servername/piwebapi/search",
                "System": "https://servername/piwebapi/system"
            }
        }
    },
    "SubRequestID2": {
        "Status": 200,
        "Headers": {
            "Content-Type": "application/json; charset=utf-8"
        },
        "Content": {
            "Items": [
                {
                    "WebId": "F1RSiOPKGTTUcUCFkVGxiCya9gUElTUlYwMQ",
                    "Name": "PISRV01"
                }
            ]
        }
    }
}
```

6.2 Parent-Child requests

One potential issue when bundling our requests is that we may need the result from one request before we can full define another request. For example, we might need to use the WebId returned from request A in the URL for request B. PI Web API provides a mechanism for this type of request, so that we can have specify some requests to execute before others and use their results in subsequent requests.

To specify that a sub-request must execute after another, we can specify the Id of the parent request in the child request’s “ParentIds” array. We can also specify an array of “Parameters”, that reference results of previously executed requests. To specify the parameter value, we use a syntax called JSONPath.

JSONPath gives a way to use the structure of unique names and arrays in JSON to define the notation of a path. As with a file path in Windows, JSONPath gives us ways to access the various sub-structures in any JSON objects.

Let’s look again at a JSON object.

```
{
    "A": 1234,
    "B": [
            {"Val": 12},
            {"Val": 34}
    ],
    "C": {
        "D": 5678
    }
}
```

To be able to define a path, we first need an identifier for the whole JSON object. This is the "$" character (think of a drive letter for a Windows path, or "/" for a Mac path). To refer to a child element, we use the dot notation. For example "$.A" refers to the value 1234.

We can refer similarly to children of children, for example "$.C.D" is equal to 5678.

For arrays, we use typical index notation, for example $.B[0].Val refers to the value 12. In the case of JSONPath, the first item is the list is access using the index 0.

The last path definition that we need, is a way to retrieve all objects within an array, we can do so with the character "*". For example, $.B[*].Val refers to an array with value 12, 34.

An example of a parent-child batch request is below. We reference the WebId returned in the request "GetId" and then use it in the child request "GetValues".

```
{
  "GetId": {
    "Method": "GET",
    "Resource":
"https://servername/piwebapi/attributes?path=\\server\\mydb\\pump|flow"
  },
  "GetValues": {
    "Method": "GET",
    "Resource": "https://servername/piwebapi/streams/{0}/recorded",
    "ParentIds": [
                 "GetId"
         ],
    "Parameters": [
                 "$.GetId.Content.WebId"
         ]
  }
}
```

The {0} in the "Resource" for the second request, tells PI Web API to insert the value of the first parameter. If there were addition parameters, they would be referenced as {1}, {2}, {3}, etc.

Exercise 6a – Get System Endpoints

1. On PICLIENT01, open the file **Exercise 6a – Get System Endpoints.py** in Visual Studio Code.

2. Build out a batch request body to return each of the endpoints under the System controller by explicitly defining each request.

3. Change the batch request body to
    a. Get the default System controller endpoint
    b. Define the “Resource” fields of subsequent requests based on the values returned in the “Links” array in the first response

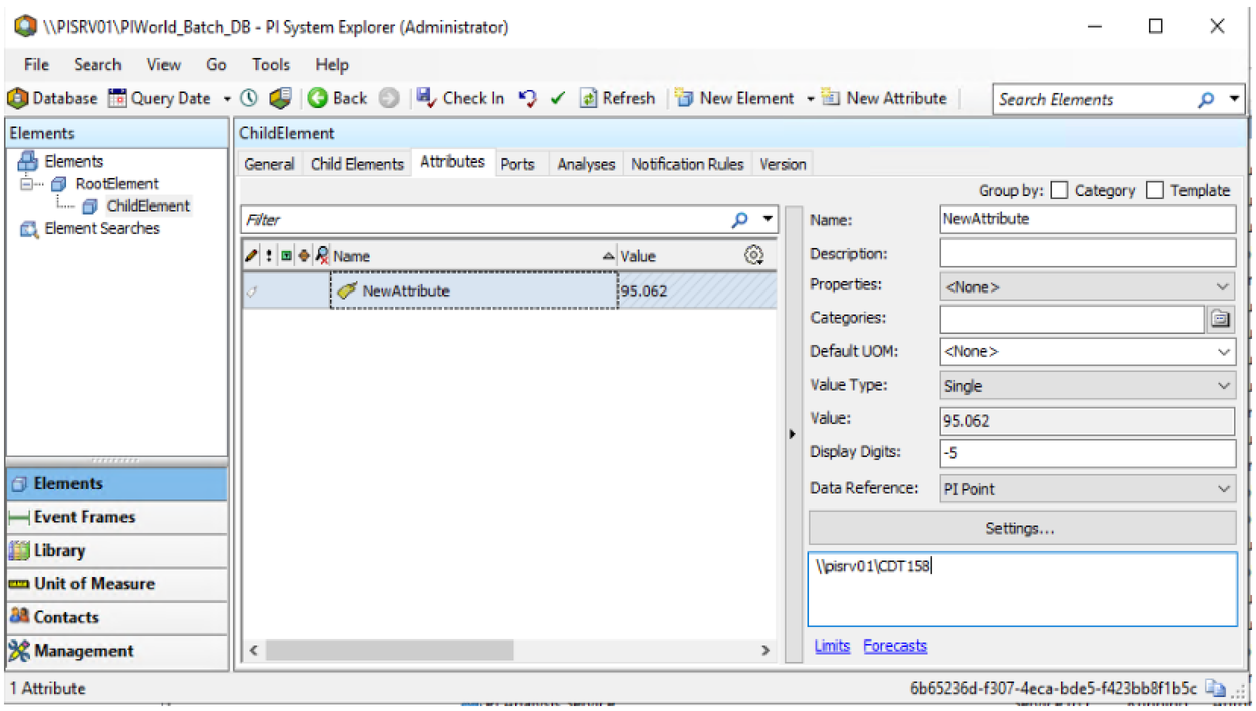
BONUS: Convert the individual child requests from part 3, into a group defined by a “RequestTemplate” where the “Resource” field is specified by a parameter.

HINT: For an example, see request ID “8” in the documentation for the Batch controller’s Execute method (link)

Exercise 6b – Create AF Hierarchy

1) On PICLIENT01, open the file **Exercise 6b – Create AF Hierarchy.py** in Visual Studio Code.

2) Edit the script to build out a batch request that creates an AF hierarchy that looks like the following:


\\PISRV01\PIWorld_Batch_DB - PI System Explorer (Administrator)
File Search View Go Tools Help
Database Query Date Back Check In Refresh New Element New Attribute
Search Elements
Elements
Elements
RootElement
ChildElement
Element Searches
Elements
Event Frames
Library
Unit of Measure
Contacts
Management
ChildElement
General Child Elements Attributes Ports Analyses Notification Rules Version
Group by: Category Template
Filter
Name Value
NewAttribute 95.062
Settings...
Limits Forecasts
1 Attribute
6b65236d-f307-4eca-bde5-f423bb8f1b5c

Name: NewAttribute
Description:
Properties: <None>
Categories:
Default UOM: <None>
Value Type: Single
Value: 95.062
Display Digits: -5
Data Reference: PI Point
\\pisrv01\CDT158

The individual steps are:

a) Get the WebId for the AF server **PISRV01**
b) Create an AF database named **PIWorld_Batch_DB**
c) Create a root element called **RootElement**
d) Create a child element under that called **ChildElement**
e) Create an PI Point Data Reference AF attribute named **NewAttribute**
f) Get the WebId for NewAttribute
g) Retrieve recorded values for NewAttribute over the last 15 minutes

HINT: If the request partially succeeds, but fails after the database was created, you will need to delete the AF database before trying again. You can do this with Postman or with PI System Explorer.

7. Channels

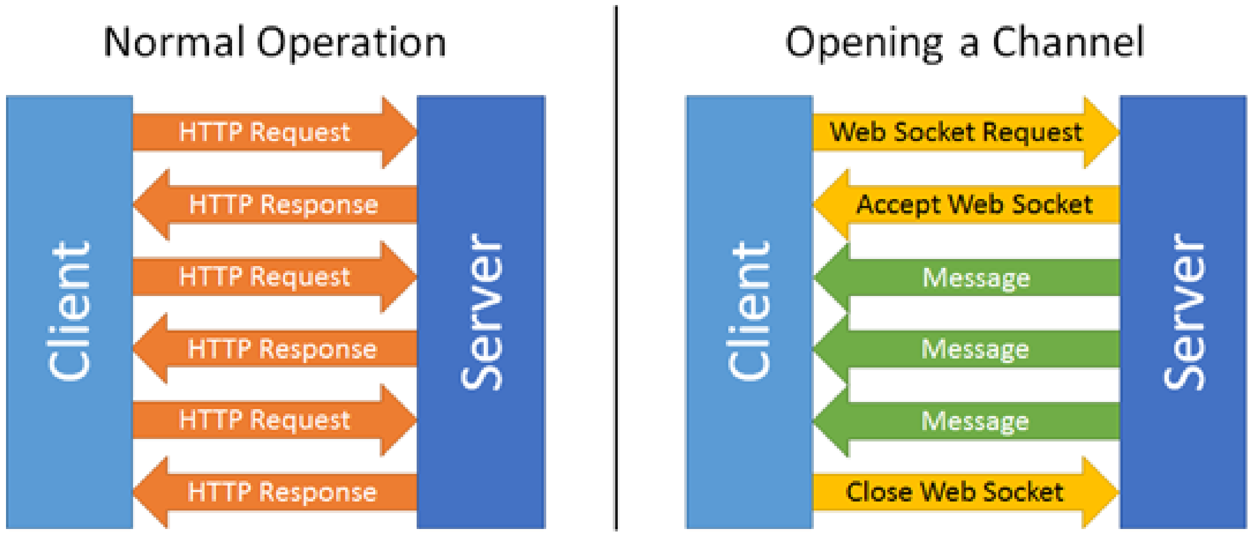
So far, we have seen how the PI Web API can be used to query for data at a point in time. However, many applications need to be able to observe real-time data and be notified when new events occur. Up to this point, the methods that we've explored would require polling the same endpoints with periodic requests to check for new data.

The Channels feature is designed for just this purpose. It allows a client to sign-up and receive continuous updates about a stream or stream set.

7.1 WebSockets

Channels are unique within PI Web API in that they are implemented using the Web Socket protocol rather than typical HTTP requests. Many programming languages have libraries available that implement the WebSocket protocol and can be used with Channels. To specify the secure WebSockets protocol the URL will begin with wss:// where we would normally see https://.

Web Sockets are different from HTTP requests in that a persistent connection is maintained between the client and the server. This allows the server to initiate the sending of messages to the client at any point while the socket is established, without having to wait for a client request.



7.2 Query Parameters

When signing up for a Channel, there are two optional query parameters that can be specified:

- **includeInitialValues** (default false) – If included and set to true, the Web API will return the current value upon establishing the initial connection
- **heartbeatRate** (default never) – If included, the web socket will send an empty message periodically, at the interval specified by this parameter

Exercise 7 – Using Channels

1) On PICLIENT01, open the file **Exercise 7 – Using Channels.py** in Visual Studio Code. This file contains the skeleton code for creating and monitoring a WebSocket.

2) Create a channel for the PI tag **CDT158**
   a. See the values come through the Channel
   b. Using either Postman or PI System Management Tools, write a value to this stream and see that event come through the channel

3) Create a channel for a stream set defined as streams under the element: **\\NuGreen\NuGreen\Tucson\Distilling Process\Equipment\B-117**

BONUS: While listening to a channel, use any of the tools we have used to this point to make a request to show the existing channel instances. How many messages have been sent for the current channel?

8. Stream updates

The Stream Updates feature allows for similar functionality as channels but using only HTTP requests. Unlike channels, stream updates require the client to actively reach out to retrieve the updates, but it avoids the need to use WebSockets.

8.1 Registering for updates

For PI Web API to start monitoring for changes on a particular stream, a POST request must be made to register the stream. This can be done for one stream at a time, or it can be done for multiple streams by using the StreamSet controller.

Example registration requests are shown below:

POST https://myserver/piwebapi/streams/{webId}/updates - Registering one stream

POST https://myserver/piwebapi/streamsets/updates?{webIds} - Registering multiple

8.2 Retrieving updates

Once registered, clients can then make requests to retrieve updates. The point at which updates are retrieved from is determined by a marker. The initial marker is returned when stream is first registered. That marker is then passed in the URL of the request, and then the PI Web API includes the next marker in the response body.

This provides multiple performance improvements over polling, because the PI Web API already has the results being queued up for when the next retrieval request comes. It also ensures that the client only receives new data and does not have to filter out data that was already returned previously.

Example retrieval requests are shown below:

GET https://myserver/piwebapi/streams/updates/{marker} - Retrieving updates using a marker

GET https://myserver/piwebapi/streamsets/updates?{markers} - Retrieving updates from multiple streams

Exercise 8 – Using Stream Updates

1) On PICLIENT01, open the file **Exercise 8 – Using Stream Updates.py** in Visual Studio Code. This contains the skeleton code for registering and retrieving stream updates

2) Complete the missing sections in order to register/retrieve Stream Updates on the tag **CDT158.**

3) Run the script to retrieve updates and see that updates are retrieved for this tag. If values are coming in too slowly, manually write some values to the tag between two retrieval requests (using Postman or PI System Management Tools).

Bonus: Change the code to use the location header in each response for the URL in the next request, rather than constructing it using the “LatestMarker”.

Bonus: Change the code to register/retrieve Stream Updates on a stream set consisting of the tags **sinusoid** and **sinusoidu** using RegisterStreamSetUpdates and RetrieveStreamSetUpdates.

9. OSIsoft GitHub

OSIsoft has a GitHub site with sample applications located at:
https://github.com/osisoft/OSI-Samples-PI-System/tree/master/piwebapi_samples

This site has examples that utilize PI Web API from various languages/frameworks. These projects implement some basic functionality and a good place to start if you are curious what an application could look like in one of the provided languages.

Final Exercise

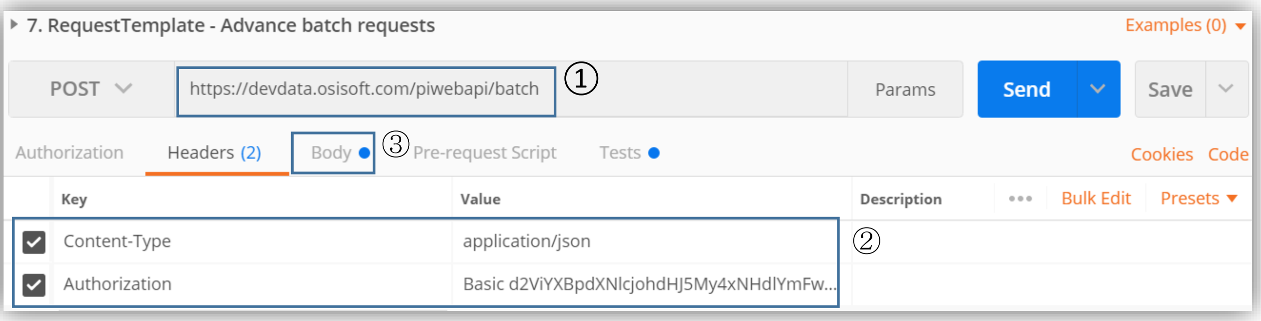
1) Download one of the example applications from GitHub
2) Follow the README.md file to install any needed pre-requisites
3) Run the application and investigate the functionality
4) Make some changes to the application to change or add features

Appendix

Postman Introduction

Postman (https://www.getpostman.com) is a free, publicly available, and easy to use tool for making HTTP requests. The UI is intuitive but also provides powerful functionality. The top half of the right pane where you will write requests against PI Web API.

Example of a request:



7. RequestTemplate - Advance batch requests

POST https://devdata.osisoft.com/piwebapi/batch ①

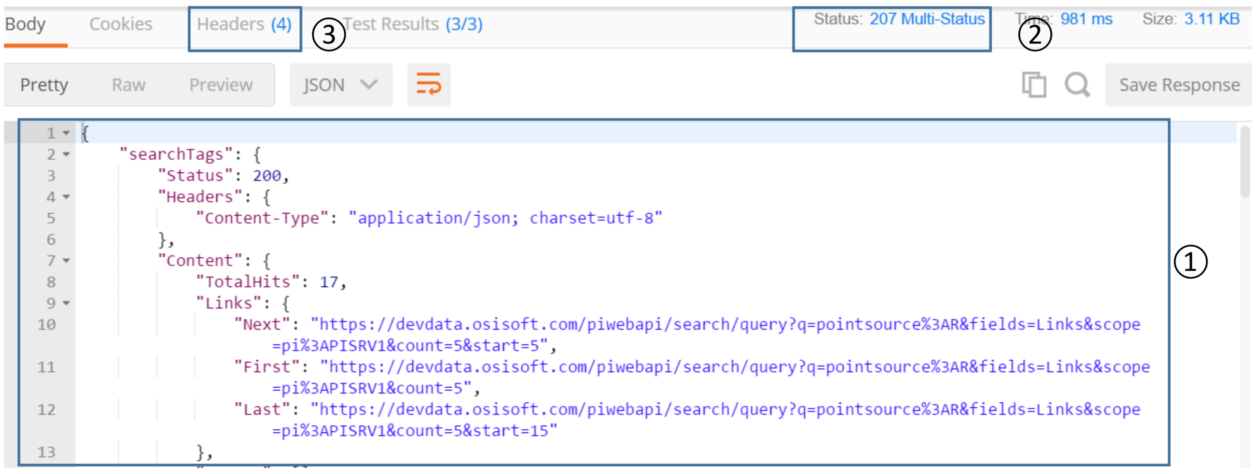
Headers (2) ②

Body ③

Key	Value	Description
✓ Content-Type	application/json	
✓ Authorization	Basic d2ViYXBpdXNlcjohdHJ5My4xNHdlYmFw...	

From this pane, you can write the URL (①) that will used in the request, fill in the header information (②) and if required the body information (③) as well. The send button will send this request to the PI Web API Service.

After sending a request to PI Web API, a response will be received. This response might include the data you requested (①) or some details about an issue your request had. The bottom half of the right pane is this response. The most important part is the body of the response when you are requesting information, but the status (②) and headers (③) will also tell you information about your request or the impact (element creation, etc.) that it had on the targeted PI System.



Headers (4) ③

Status: 207 Multi-Status ② Time: 981 ms Size: 3.11 KB

```
{
    "searchTags": {
        "Status": 200,
        "Headers": {
            "Content-Type": "application/json; charset=utf-8"
        },
        "Content": {
            "TotalHits": 17,
            "Links": {
                "Next": "https://devdata.osisoft.com/piwebapi/search/query?q=pointsource%3AR&fields=Links&scope
                    =pi%3APISRV1&count=5&start=5",
                "First": "https://devdata.osisoft.com/piwebapi/search/query?q=pointsource%3AR&fields=Links&scope
                    =pi%3APISRV1&count=5",
                "Last": "https://devdata.osisoft.com/piwebapi/search/query?q=pointsource%3AR&fields=Links&scope
                    =pi%3APISRV1&count=5&start=15"
            },
```
①

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

OSIsoft. Learning
PI SYSTEM LEARNING MADE EASY!
Accelerate success with the new OSIsoft Learning platform.
VISIT LEARNING.OSISOFT.COM