Developing with PI Web API

PI DEVELOPERS CLUB WHITE PAPER

OSISOFT, LLC

How to Contact Us

Email: pidevclub@osisoft.com
Web: pisquare.osisoft.com/community/developers-club

OSIsoft, LLC
777 Davis St., Suite 250
San Leandro, CA 94577 USA
Tel: +1 510-297-5800
Fax: +1 510-357-8136
Web: http://www.osisoft.com

Table of Contents

Overview

About this Document

This document is available as part of the PI Developer Club website content, under the PI Developers Club White Paper and Tutorials category, here.

Any question or comment related to this document should be posted in the appropriate PI Developers Club discussion forum or sent to the PI Developers Club Team at pidevclub@osisoft.com.

About this White Paper

PI Web API was released as the product which enables cross-platform development of web, desktop, and mobile applications across many different programming languages. This white paper will help you get started developing on the following platforms:

- ASP.NET MVC
- JavaScript/jQuery
- PHP
- Java
- Matlab
- R

What You Need to Start

We have tested the examples with the following products:

- PI Web API 2015 R2
- PI Data Archive 2015
- PI AF Server 2015
- Google Chrome
- Fiddler Web Debugger 4.4.9.4
- Visual Studio 2012/2013/2015
- Eclipse Kepler SR2 for PHP and Java (optional)
- Matlab 2014 (optional)
- R 2.15.3 (optional)

Although there is no guarantee, recent versions from the products above shall work as well.

1. Introduction to PI Web API

What is PI Web API?

PI Web API is a RESTful PI System access layer that provides a cross-platform programmatic interface to the PI System. RESTful interfaces enable cross-platform development of web, desktop, and mobile applications across many different programming languages. PI Web API enables you to retrieve and manipulate time series data from the PI Data Archive, and asset and event frame data from the PI AF server.

What is a Web Service?

A Web Service is a method of communication provided at a network address over the web. It describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL standards over an Internet. It is commonly used as a means for businesses to communicate with each other and with clients, as it allow organizations to communicate data without taking into account different IT infrastructures.

SOAP

SOAP is an XML-based messaging protocol. It defines a set of rules for structuring messages that can be used for simple one-way messaging but is particularly useful for performing RPC-style (Remote Procedure Call) request-response dialogues. It is not tied to any particular transport protocol though HTTP is popular. This type of web service supports several protocols and technologies, including WSDL, XSDs, and WS-Addressing. SOAP request and response data are always structured and defined by a schema.

REST

In REST architecture there is always a client and a server where the communication is always initiated by the client. The client and server are decoupled by a uniform interface thereby making both the client and server to develop independently. Every resource in the server is accessed by a unique address (URI). When the client access a resource the server returns a representation of the resource based upon the request header. The representations are usually called as media-types or MIME types. A very common media-type is JSON which is used by PI Web API.

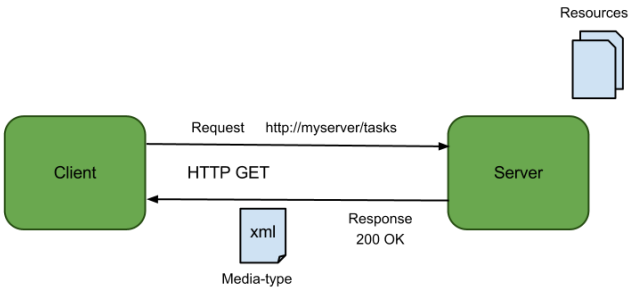


Figure 1 – Client and Server communication in a REST architecture.

What is the advantage of web services over other kinds of data access?

One advantage of using Web Services is that as almost all the communication is in XML or JSON, which means that it is not tied to any operating system or programming language. Therefore, a PHP application can talk to other applications developed in C#, Java or C++ among others that support HTTP. As such it is an important building block for developing distributed applications that exploit functionality published as services over an intranet or the internet.

Concerning the client application that are consuming the web service, no local installation are required. What is important is that the application supports HTTP.

Finally their concepts are easier to learn when compared to other types of data access such as PI OLEDB Enterprise, PI ODBC, or PI AF SDK.

Web Services in the Industry

Representational State Transfer (REST) has gained widespread acceptance across the Web as a simpler alternative to SOAP. The key evidence of this shift in interface design is the adoption of REST by mainstream Web 2.0 service providers—including Yahoo, Google, and Facebook who have deprecated or not implemented SOAP and WSDL-based interfaces in favor of an easier-to-use, resource-oriented model to expose their services.

Common Scenarios for PI Web API

There are some scenarios suitable for using PI Web API as a back end:

- **Rich-client web applications**: PI Web API is a good option for rich web applications that heavily use AJAX to get to a business or data tier. Single-page applications which are built using JavaScript libraries as jQuery, AngularJS, Backbone, Knockout, are becoming popular and they are fully compatible with PI Web API.
- **Native mobile apps**: PI Web API is also a great fit for mobile native applications for Android, iPhone or Windows Phone. All three related SDKs make it easy to take advantage of RESTful web services.
- **Cross-platform apps:** A lot of different platforms and programming languages support PI Web API. This means that with this product, it is easy to develop an app that gets PI System data from Linux.
- **Windows apps:** An advantage of developing an app on top of PI Web API is that it does not require any OSIsoft product to be installed on every client machine. This is not true for an application developed using PI AF SDK, where PI AF Client needs to be installed as a prerequisite for running the application.

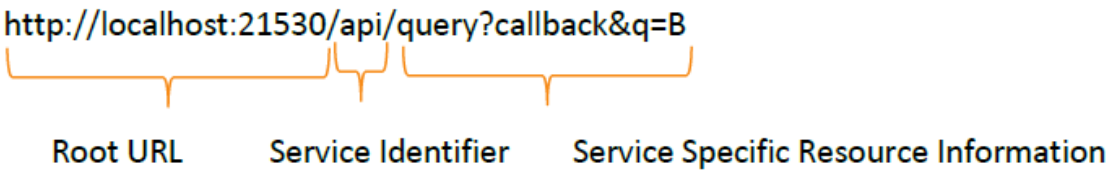
2. Working with PI Web API

Interacting with RESTful Services

PI Web API is exposed by a series of backend services. In order to interact with a service you'll write code that sends HTTP requests comprised of the following basic building blocks:

Root URL + Service Identifier + Service Specific Resource Information

Let's look at an example to define these components.



Root URL is deployment environment specific.

Service Identifier – In the example above, the *api* service is specified as the Service Identifier. For PI Web API, the service identifier is *piwebapi* by default.

Service Specific Resource Information – Developing a REST service means thinking in resources, HTTP verbs, and hypermedia controls. What resources will your service expose, what are their behaviors, and what are the syntax and semantics associated with related interactions? For the suite of REST services, you need to have access to its API documentation.

HTTP Verbs

PI Web API is an Application Programming Interface that allows you to manipulate PI System data over HTTP requests.

But what an HTTP request is? Whenever your web browser fetches a resource from a web server, it does so using HTTP - that's "Hypertext Transfer Protocol." HTTP is a request/response protocol, which means your computer sends a request for some file or JSON response and the web server sends back a response. There are a lot of different HTTP methods. The main ones are: GET, POST, PUT, PATCH and DELETE.

One might think that the CRUD operations create, read, update, and delete correspond to the HTTP methods POST, GET, PUT, and DELETE. Although GET could be associated with reading and DELETE with deleting, POST for creating and PUT for updating are not fully correct. Let's describe each one of them:

GET

GET is guaranteed not to cause any side effects and it is said to be nullipotent; nothing happens to the system's state, even when it is called multiple times or not called at all. In other words, the system state will be the same for all the following scenarios:

- The method was not called at all,
- The method was called once, and
- The method was called multiple times.

If the system finds the resource requested it will return returning *200 – OK*, otherwise *404 – Not Found*.

Example: Access or read an element using its webId → GET elements/{webId}

If you are not familiar with what the webId is, please refer to the upcoming section "What is WebId?"

DELETE

DELETE is idempotent which means that the effect to the system state will be the same as from the first call, even when they are called multiple times subsequently. After the first request to delete a resource, which is successful returning *200 – OK*, if there are subsequent DELETE requests to the same resource, we will have a problem in finding the resource in subsequent requests returning *404 – Not Found*.

Example: Delete an element using its webId → DELETE elements/{webId}

PUT

PUT is also idempotent. A new resource can be created by an HTTP PUT to the URI with an ID. If the resource with the specified ID does not already exist, the new resource is created (*200 – OK*). Otherwise, it will be updated returning (*204 – No Content*). The response body should contain the representation of the resource.

Partial updates are not allowed. Therefore, this type of resource creation is analogous to INSERT SQL statements that specify the primary key.

Example: Create or update a configuration item → PUT system/configuration/{key}.

POST

A new resource can also be created by an HTTP POST to the URI but without the ID. As no ID is defined, the HTTP POST will create a new resource with a unique ID. If the HTTP POST without an ID specified is called again, a new resource is created with a different and unique ID. This is the reason why POST is not idempotent.

HTTP POST can be used to update a resource in case the ID is defined the same way as HTTP PUT.

Example: Create a child element → POST elements/{webId}/elements.

PATCH

HTTP PATCH method supports partial updates to a resource. Supposing that a resource has many fields, PATCH should be used if you want to update only one of them. PUT is also able to update the resource but all fields must be present on the body request. Therefore, PATCH utilizes less bandwidth than PUT as a general rule.

Example: Update an element by replacing items in its definition → PATCH elements/{webId}

Google Chrome, Postman and Fiddler

In this white paper, Google Chrome was chosen as our browser due to the fact that not only it has great developer tools (accessed by pressing the F12 key) but also because there is a great Chrome extension named Postman, which is an outstanding HTTP client that makes it easy to test web services by manually crafting HTTP requests.

You can download Google Chrome from https://www.google.com/chrome/browser and, once it is installed, get the Postman client from http://www.getpostman.com.

An alternative to Postman is Fiddler Web Debugger. Fiddler will be used on the cache section of this chapter.

This white paper was written using Google Chrome 44.0.2403, Postman 3.0.5, Fiddler Web Debugging 4.4.9.4 and PI Web API 2015 R2.

PI Web API Help

Please refer to the PI Web API help in case you need to know the structure of the URI and the body message to make PI Web API calls. The service specific resource information for the help is: *help*. Therefore, as the URI Root with the Service Identifier is https://Web-Server-Name/piwebapi/, the help page can be found on https://Web-Server-Name/piwebapi/help.

Let's do a quick example which retrieves the content from a specific element whose webId is already known. You don't know how to generate the url.

Open the PI Web API online help and click on the Element link under Controllers section. There are a lot of actions available. As you know the webId already, the ***Get*** action should be used. If you click on this action, you will see that the format of the url is: "elements/{webId}".

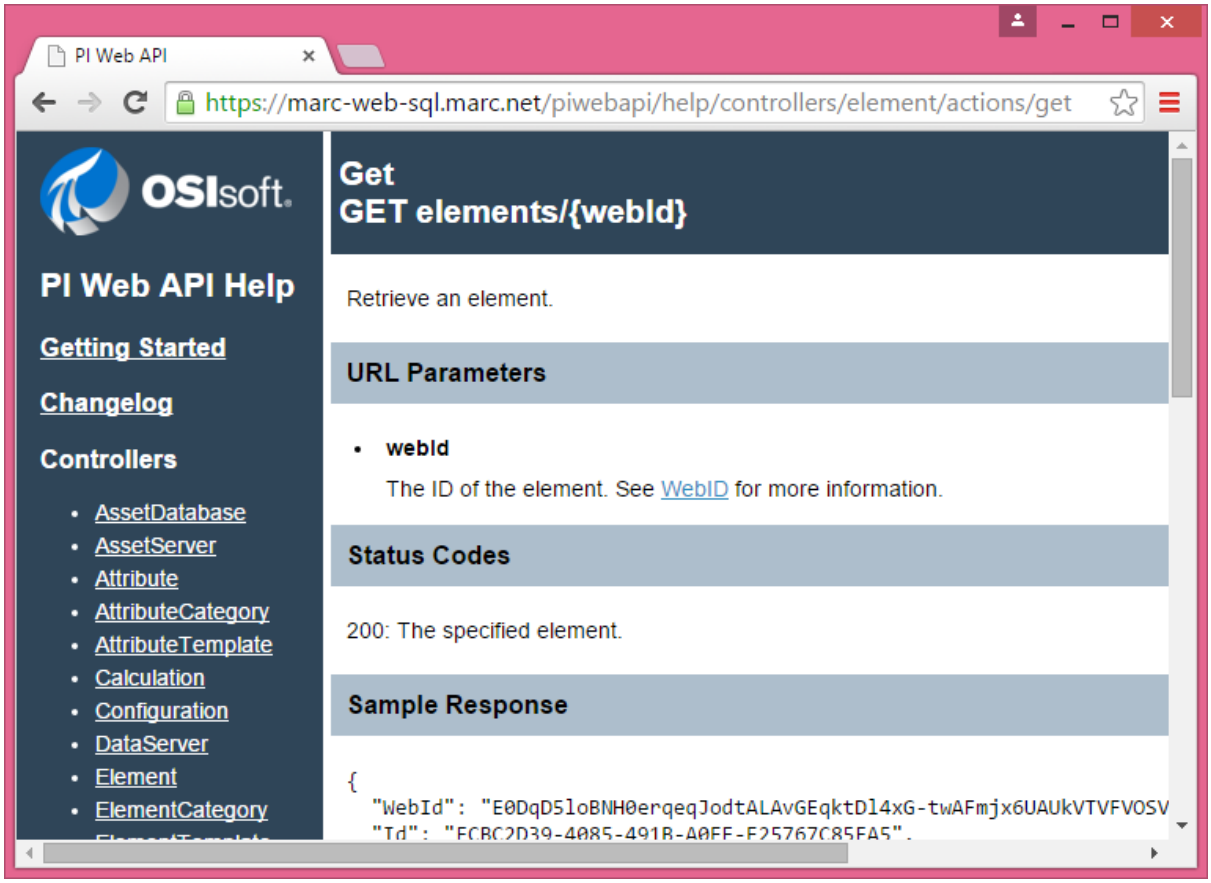


Figure 2 – Using PI Web API online help

If your webId ***E0DqD5loBNH0erqeqJodtALAvHUkVFTg*** for instance, the url that would retrieve the element information would be:
https://Web-Server-Name/piwebapi/elements/*E0DqD5loBNH0erqeqJodtALAvHUkVFTg.*

If you know only the element path, you should use the ***GetByPath*** action. An example of retrieving an element by path using query strings is:
https://Web-Server-Name/piwebapi/elements?path=\\SERVERNAME\DBNAME\Nugreen.

PI Web API Authentication

PI Web API works with three types of security: Kerberos, Basic and Anonymous. The type of security is stored on the *AuthenticationMethods* attribute of the *System Configuration* element under the Configuration database of the PI AF Server.

HTTP headers are the core part of these HTTP requests and responses, and they carry information about the client browser, the requested page, the server and more. PI Web API uses the HTTP header to carry the authentication, compression and cache information.

Anonymous Authentication

Anonymous authentication which does not require an authentication header on the request. Anyone can access the resources from PI System given the PI Web API service account allowed privileges. It is recommended to use Basic or Kerberos authentication instead of anonymous authentication. Nevertheless, if you choose this option, do not forget to set up the *DisableWrite* attribute to true when using this type of authentication. This will prevent malicious users from deleting or modifying important information from your PI System.

Basic Authentication

Basic authentication is less secure than Kerberos authentication. As Kerberos authentication requires communication between the client and the domain controller, there are certain scenarios where is not possible to use it as, for instance, accessing PI Web API from the public internet.

The login credentials are combined into a string "username: password" which is then encoded using the Base64 scheme. This encoded string is then sent using an authorization header on each request from the browser. Because the credentials are only encoded, not encrypted, this is highly insecure unless it is sent over https. This is one of the reasons why PI Web API does not work with HTTP.

The header of basic authentication is: *Authorization: Basic xxxxxxxxxxxxxxxxxx*

Kerberos Authentication

Kerberos authentication is the most secure option available on PI Web API nevertheless all the computers involved must be members of the same domain.

Setting up Kerberos authentication properly is not always an easy task. There are some OSIsoft KB articles available in order to help you:

- KB01223 - Kerberos and Internet Browsers
- KB01222 - Types of Kerberos Delegation

The header of kerberos authentication is: *Authorization: Negotiate xxxxxxxxxxxxxxxxx*

Note: If you receive a response with the status code 401 (Unauthorized) after making an HTTP request, this indicates that there was a problem authenticating to PI Web API.

Using Postman and Fiddler to make PI Web API calls

After setting up an environment with Google Chrome, Postman and PI Web API running properly, it is time to start making some HTTP calls. If you are facing any issue with PI Web API, please refer to our PI Web API Troubleshooting Guide.

PI Web API Root

Open Postman on Google Chrome and write URI Root with the Service Identifier. Make sure GET is selected for the HTTP method and no URI parameters are defined. If you are not using anonymous as one of the authentication methods, you should add a header for the Basic or Kerberos authentication. Nevertheless, the current version of Postman does not seem to be compatible with Kerberos authentication. Please set up Basic authentication if you want to follow the examples of this chapter. Click on the “Send” button.

You will receive a JSON response:


```
{
    "Links": {
        "Self": "https://marc-web-sql.marc.net/piwebapi/",
        "AssetServers": "https://marc-web-sql.marc.net/piwebapi/assetservers",
        "DataServers": "https://marc-web-sql.marc.net/piwebapi/dataservers",
        "System": "https://marc-web-sql.marc.net/piwebapi/system"
    },
    "HtmlHelp": "https://marc-web-sql.marc.net/piwebapi/help"
}
```

If you type the same URI on Google Chrome, you will actually receive an HTML page with a JSON response included. How can you receive different responses using the same URI?

Open Google Chrome developer tools (by pressing F12), go to the Network ribbon and refresh the page. Review the content of the “Request Headers” section. You shall see the following header:

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

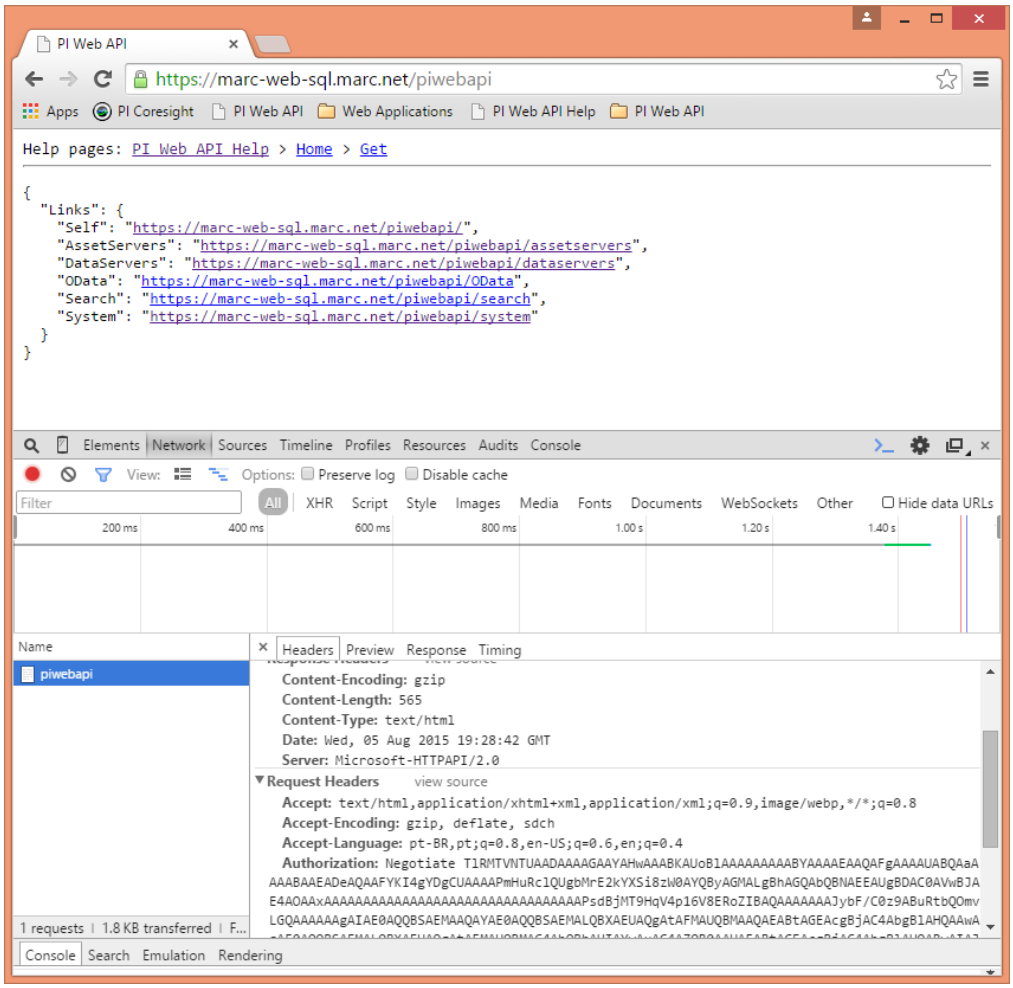


Figure 3 – Accessing PI Web API with Google Chrome

Let’s change the headers on Postman by adding the **Accept** Parameter Key and its value with the same content shown on Chrome. Then, click on the “Send” button and you should receive a string starting with “<!DOCTYPE html>” which means that this is an HTML page.

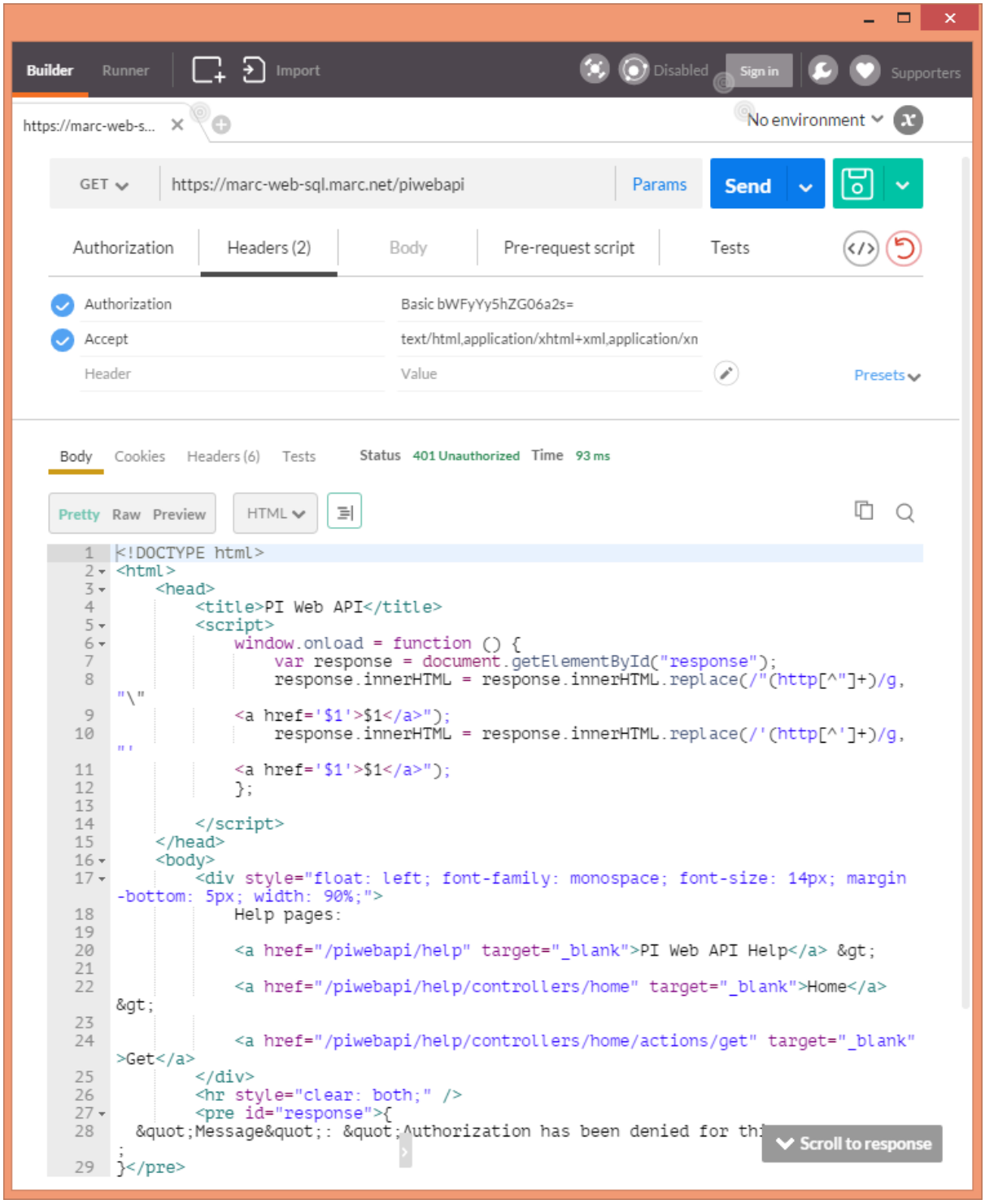


Figure 4 – Accessing PI Web API with Postman with the accept header

The conclusion is that the response is influenced not only by the URI but also by the header. Please delete the “Accept” key as the JSON response (and not the HTML response) is more useful for us to complete the samples of this chapter. To do so, select the header and just click on the X button on the right of the fields.

What is WebID?

Most of the objects returned on PI Web API responses has a field called WebID. This is the case for PI Tags, AF Attributes, AF Elements and Asset Servers. The WebID is a unique identifier that are used on PI Web API queries. It is generated by encoding the ID of the requested object and its path. When PI Web API receives an HTTP request with the WebID, it decodes the WebID and it finds the object ID and its path. Even if it does not find the object ID, it will still try to get the object using the path.

Please refer to the PI Web API User Guide for more information concerning this topic.

Accessing AssetServer and AssetDatabases

Click on the link related to AssetServers. In our case, https://marc-web-sql.marc.net/piwebapi/assetservers. Then, click on "Send." You will see the list of all PI AF Servers stored on the web server running PI Web API.


```
{
    "Links": {},
    "Items": [
        {
            "WebId": "S07dtyl5PO4EmFExFPTz6FbgTUFSQy1QSTIwMTQ",
            "Id": "9772dbed-ce93-49e0-8513-114f4f3e856e",
            "Name": "MARC-PI2014",
            "Description": "<Not Connected>",
            "Path": "\\\\MARC-PI2014",
            "ServerVersion": "",
            "Links": {
                "Self": "https://marc-web-
sql.marc.net/piwebapi/assetservers/S07dtyl5PO4EmFExFPTz6FbgTUFSQy1QSTIwMTQ",
                "Databases": "https://marc-web-
sql.marc.net/piwebapi/assetservers/S07dtyl5PO4EmFExFPTz6FbgTUFSQy1QSTIwMTQ/assetdatabas
es"
            }
        },
        {
            "WebId": "S0IoUwPkYie0icb7l0KWXb6gT1NJLVNFUlY",
            "Id": "3e308522-2246-487b-9c6f-b9742965dbea",
            "Name": "OSI-SERV",
            "Description": "<Not Connected>",
            "Path": "\\\\OSI-SERV",
            "ServerVersion": "",
            "Links": {
                "Self": "https://marc-web-
sql.marc.net/piwebapi/assetservers/S0IoUwPkYie0icb7l0KWXb6gT1NJLVNFUlY",
```

```
                "Databases": "https://marc-web-
sql.marc.net/piwebapi/assetservers/S0IoUwPkYie0icb7l0KWXb6gT1NJLVNFUlY/assetdatabases"
            }
        }
    ]}
```

If you want to get information from a specific PI AF Server, there are three other options:

- Filtering by name: …./piwebapi/assetservers?name=AFSERVERNAME
- Filtering by path: …./piwebapi/assetservers?path=\\AFSERVERNAME
- Clicking on the link of the “Self” of the selected PI AF Server.

On links array, click on the link to view all the AF databases from the selected AF Server.

Create a new root element from the database

There are also GET methods to retrieve AssetDatabase information by WebID or by path. But if you read the documentation, there is an option to create an element or element template at the AF database root. This is what we are going to do.

To create new elements, we need to use the following structure with HTTP POST request: assetdatabases/{webIdOfTheAssetDatabase}/elements where WebID is related to the AssetDatabase. As no categories are specified and as this element is not derived from any element template, this is the request body:

```
{
  "Name": "ElementWhitePaper",
  "Description": "Element created by the White Paper"
}
```

On Postman, type the correct URI and select the HTTP POST method. On the header, make sure to add a key “Content-Type” whose value is “application/json.” Then choose the “raw” option and paste the requested body described above. Click on “Send” to create a new element.

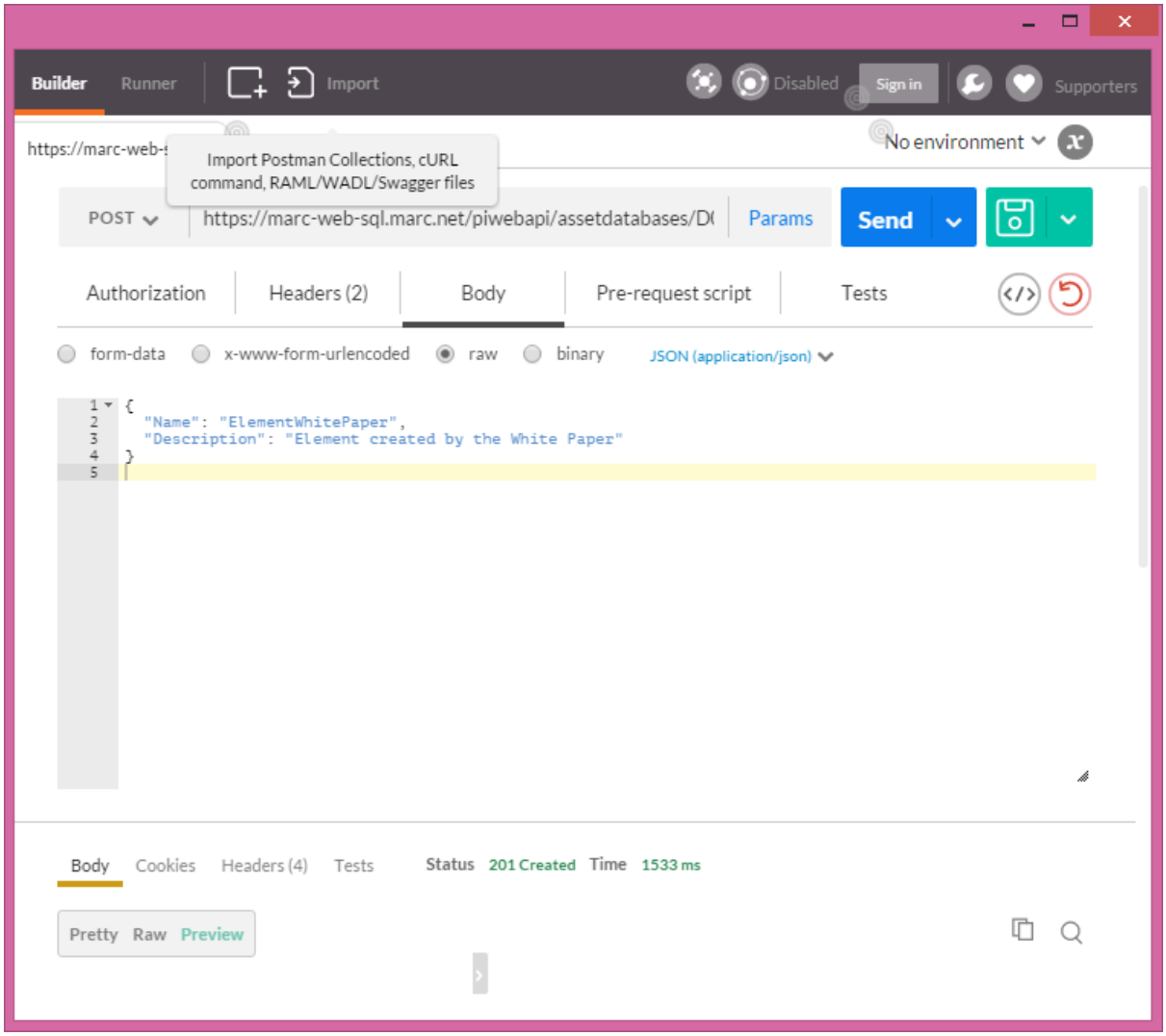


Figure 5 – Creating a new element root with POSTMAN

If the request executes successfully, Postman receives a *201 – Created* which means that the element was created successfully as shown on Figure 4.

Create a child element on a parent element

Using the same WebID from the assetdatabases, let's make another request with the same URI but this time a GET request. This will result on all the root elements from the selected database. You will have access to the WebID from the new element ElementWhitePaper just created.

With the new WebID from the new element created, we will create a new child element called ChildElementWhitePaper. Make a similar POST request with the following URI structure: https://Web-Server-Name/piwebapi/elements/{webIdElementCreated}/elements.

Change the name and description from the body request as shown on Figure 6:

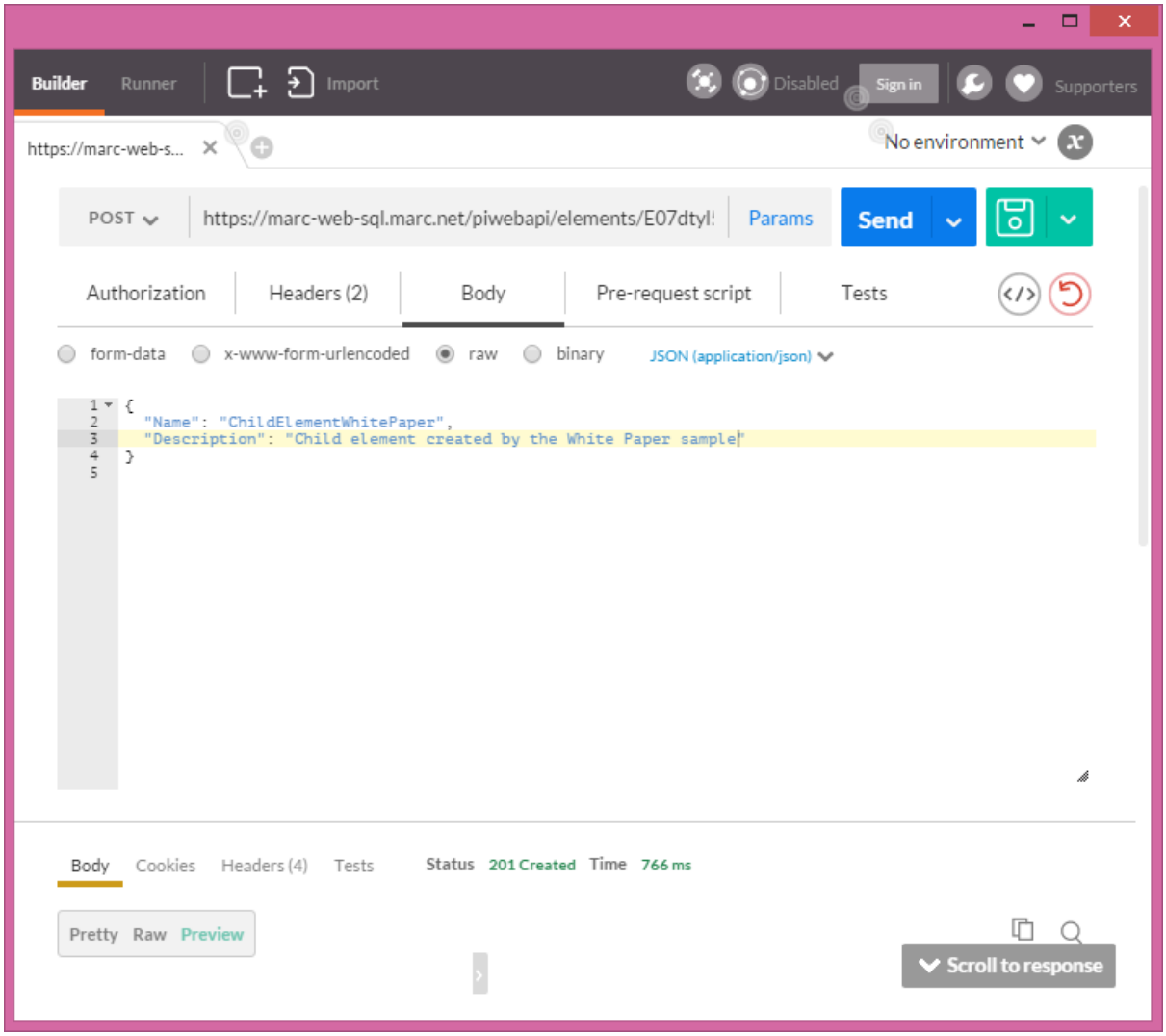


Figure 6 – Creating a new child element

Create a new attribute

Creating a new attribute is very similar to creating a new element. We are going to create the attribute called "AttributeWhitePaper" using the same WebID from the previous section. Make a similar POST request changing the URI to:
https://Web-Server-Name/piwebapi/elements/{webIdElementCreated}/attributes

Please refer to the PI Web API help in order to find out the structure of the body request. In this example please use the following body message:

```
{
  "Name": "AttributeWhitePaper",
  "Description": "Attribute Created by the White Paper",
  "DefaultUnitsName": "meter",
  "DataReferencePlugIn": "PI Point",
  "ConfigString": "\\\\MARC-PI2014\\CDT158;UOM=m;ReadOnly=False"
}
```

This will create a new attribute called AttributeWhitePaper with a description, using the PI Point Data Reference pointing to the CDT158 pi tag. It uses the meter as the default unit of measure. Click on the send button to create the attribute. You should receive a 201 status code as well as shown on Figure 7.

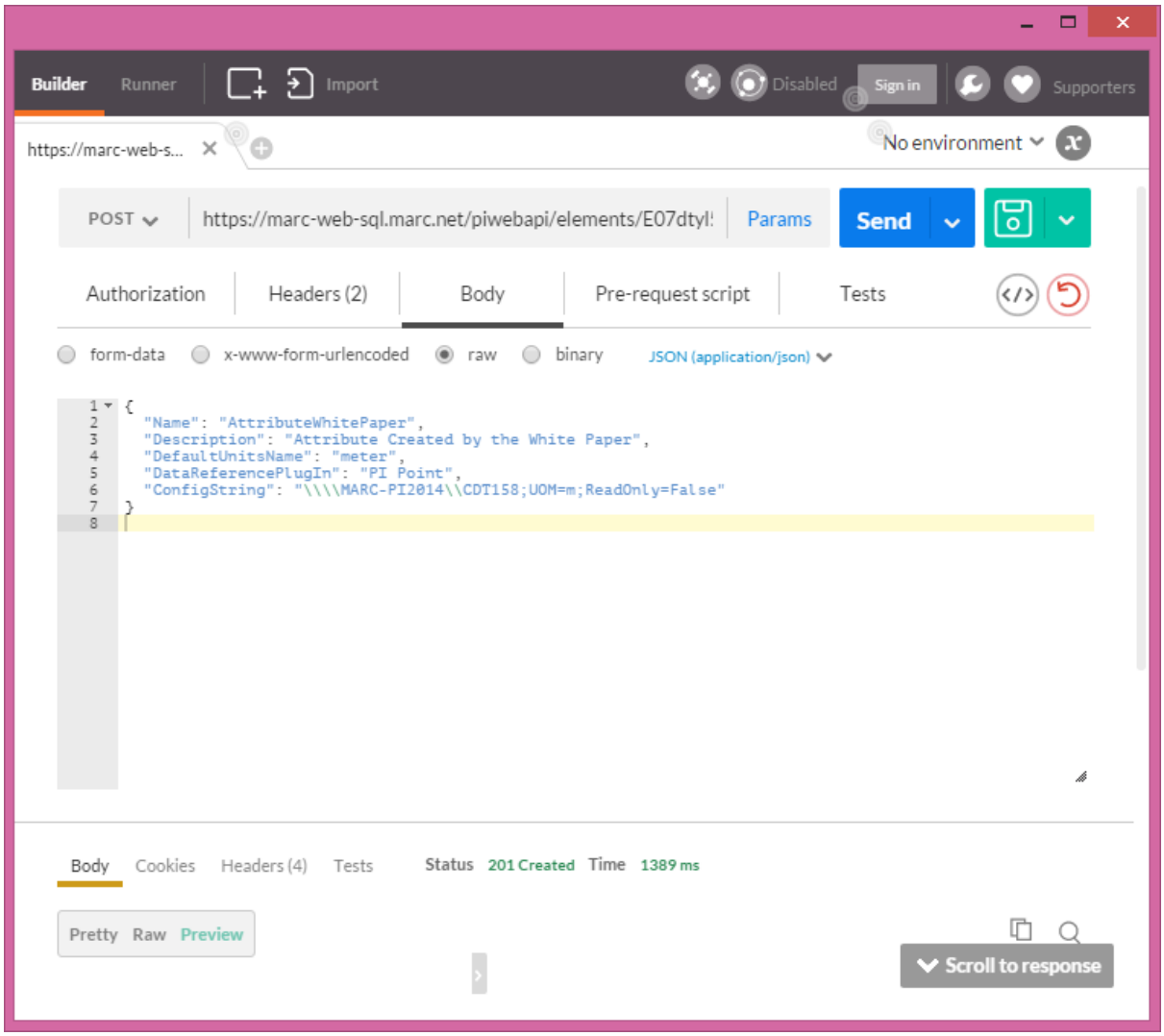


Figure 7 – Creating a new attribute

Update an attribute

Let's suppose the PI Point of this new attribute needs to be changed from "cdt158" to "sinusoid." As only some fields of the attribute properties must be updated, PATCH request is the best choice in order to utilize less bandwidth.

The new attribute's WebID needs to be found first to make the update request. Using the same URI from the previous section, change the request method to GET and click on the Send button. You will see the attributes information of the ElementWhitePaper.

Using the WebID from the attribute taken from the GET response, change the method to PATCH and the URI to:

https://Web-Server-Name/piwebapi/attributes/{webIdAttributeCreated}/

As only the ConfigString should be updated, the following body request will be used:

```
{
  "ConfigString": "\\\\MARC-PI2014\\SINUSOID;UOM=m;ReadOnly=False"
}
```

Click on the “Send” button and if it works successfully, the 204 status code will be received (and not 201). Please refer to Figure 8.

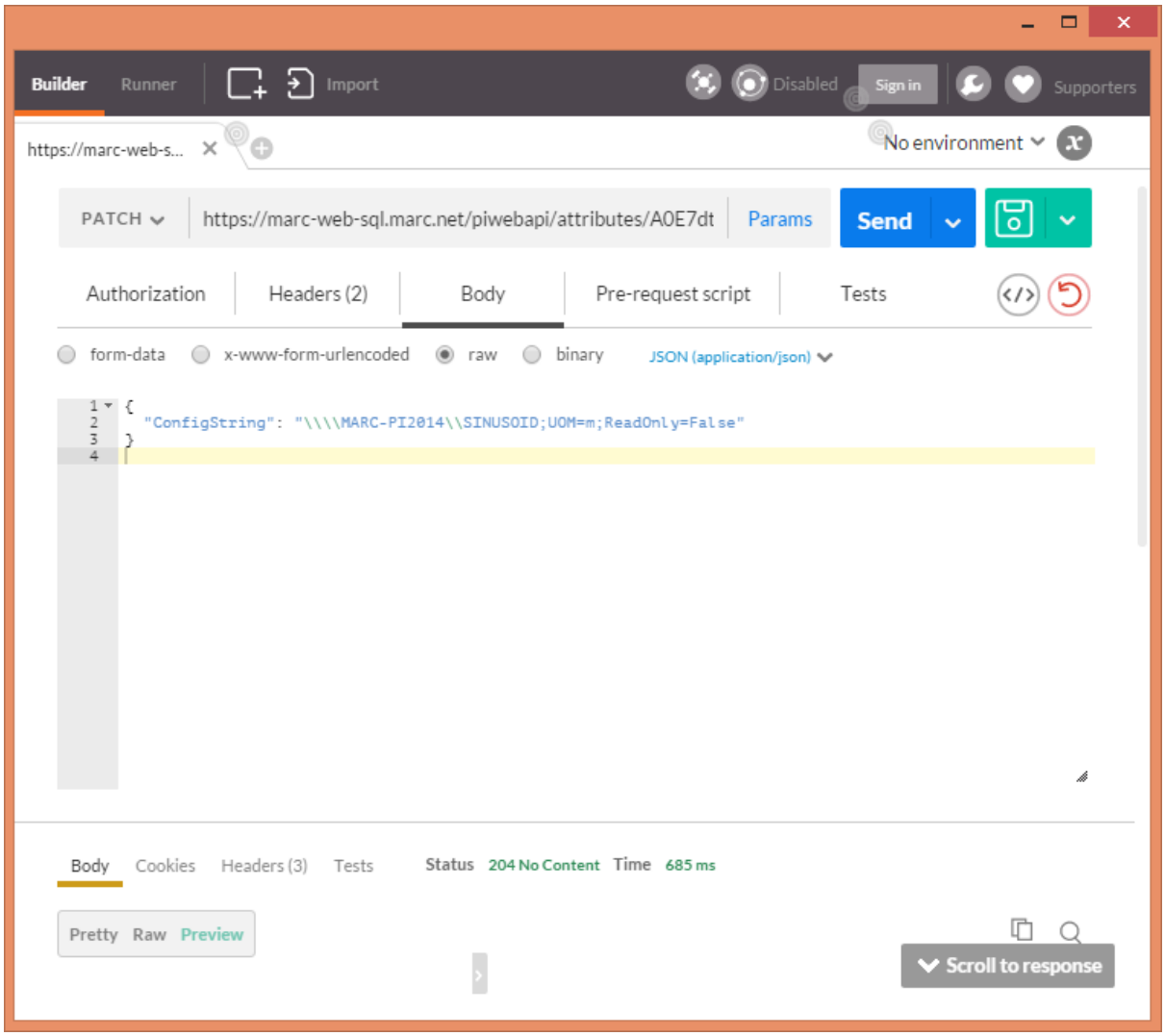


Figure 8 – Updating an attribute

Update the attribute's value

Change the method to GET and click on send. Within the "Links" array on the JSON response, you will find a link associated to the value whose structure is:

https://Web-Server-Name/piwebapi/streams/{webId}/value

Figure 9 shows how the URI should be extracted from the JSON response.

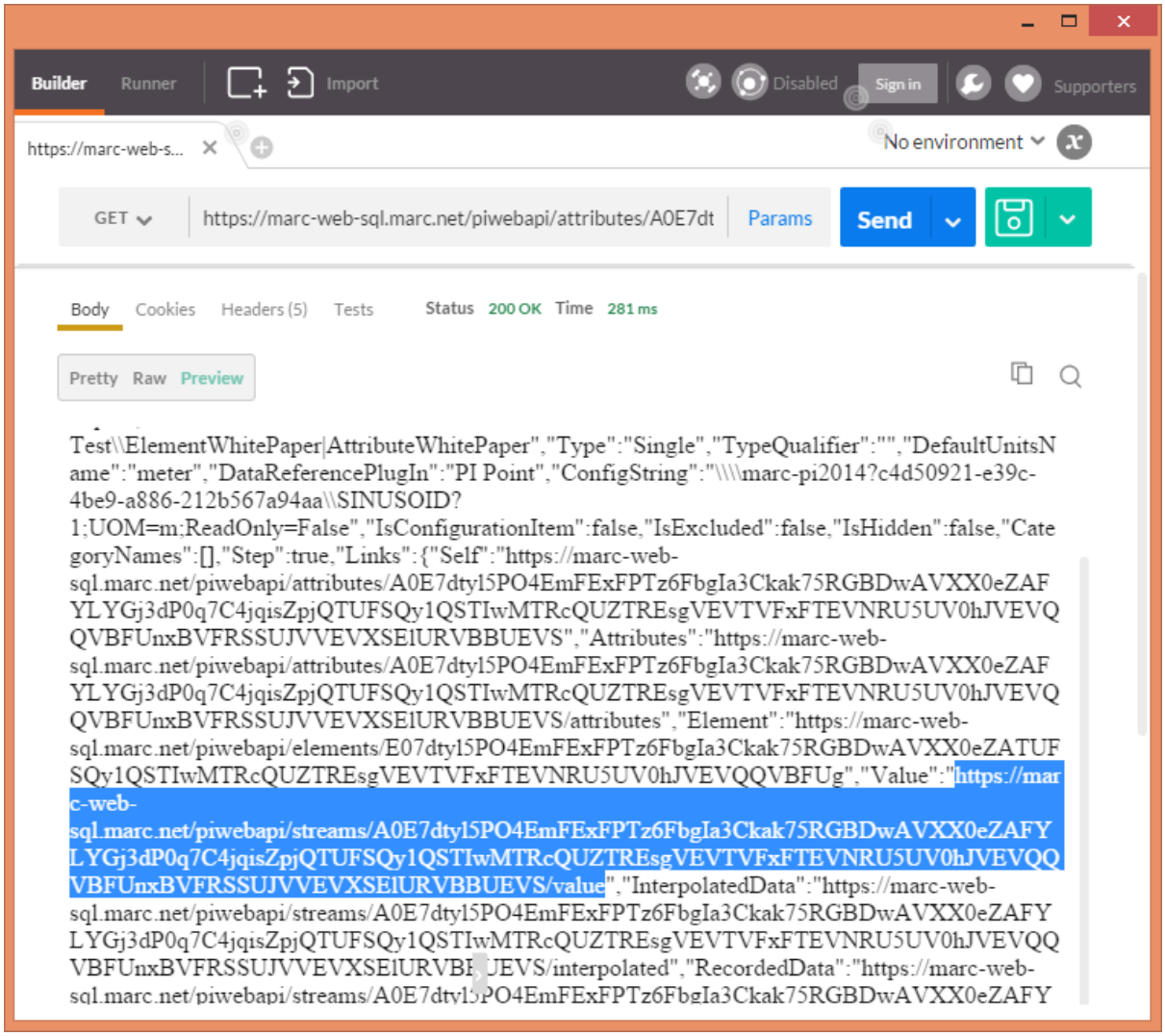


Figure 9 – Getting the URL to update the snapshot

With the URI, changing the request method to POST, and using body message below, the snapshot is updated to 200 as shown on Figure 10:

```
{
  "Value": 200
}
```

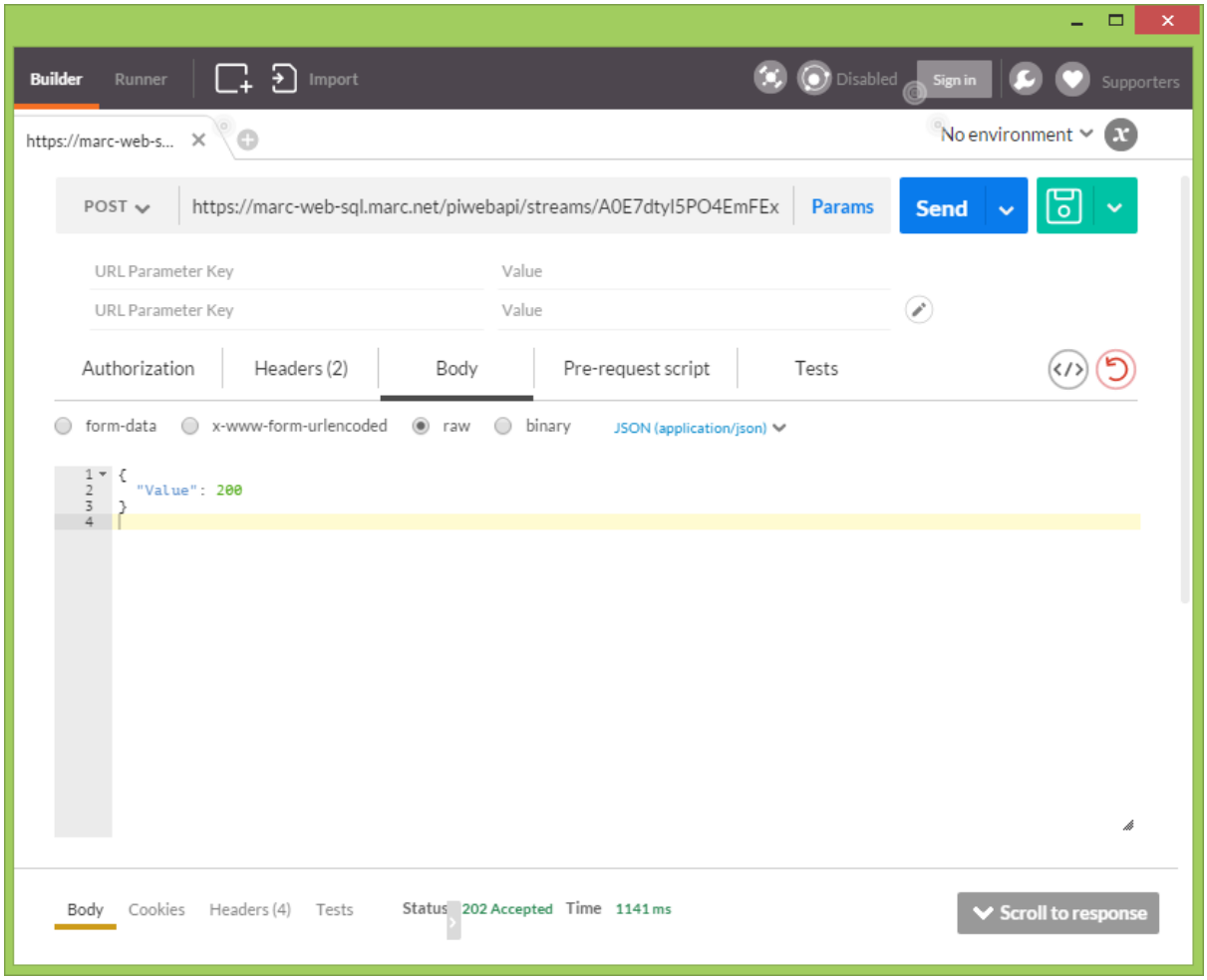


Figure 10 – Updating the snapshot

Using PI Web API Cache

As PI Web API is developed on top of PI AF SDK, it takes advantage of PI AF SDK cache when retrieving AF objects. PI Web API allocates a separate cache for each user. For more information about the PI Web API cache please refer to the "PI Web API caching" chapter of the "PI Web API 2015 R2 User Guide."

It is possible to perform this section using the current version of Postman but it can be a little tricky as Postman might change the cache header automatically under the hood. In order to make it work properly, open Postman preferences and under the Settings ribbon, change "Send no-cache header" to "No."

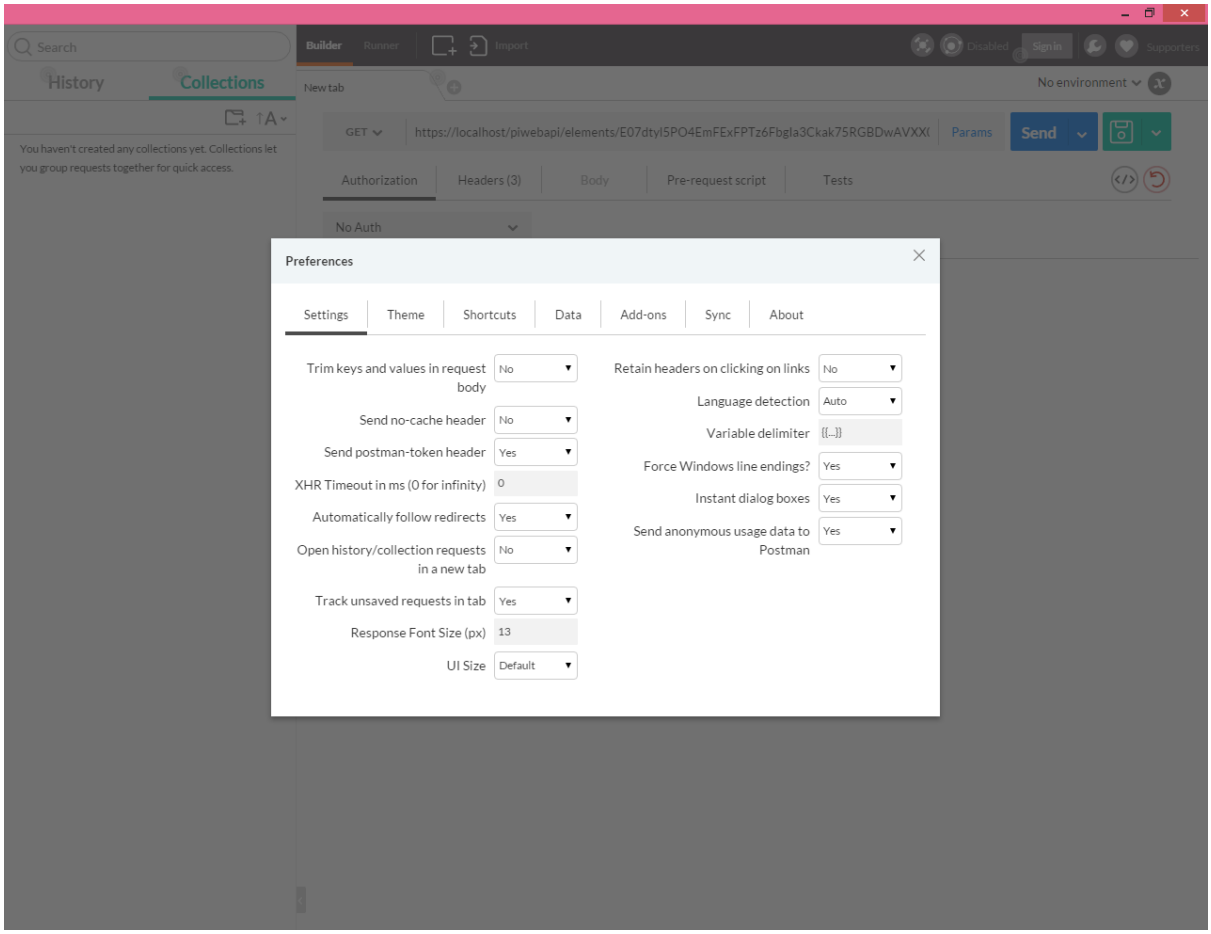


Figure 11– Editing Postman preferences

On this section, Fiddler will be used instead in order not only to have full control of the requests headers, but also to provide a second alternative to test you PI Web API HTTP requests.

On the previous section, the attribute called *AttributeWhitePaper* was created with a description. Open Fiddler and get this attribute properties with the cache header to request the latest version of this object.

After finding the attribute WebID, you can use the following endpoint service: attributes/{webId}. Finally, add the following cache header to the request: "Cache-Control: max-age=0." Execute the request and make sure you get the expected response with the attribute properties.

Open PI System Explorer and find this attribute. Then, change its description to: "Changed description attribute" as shown on Figure 12. Make sure to check in.

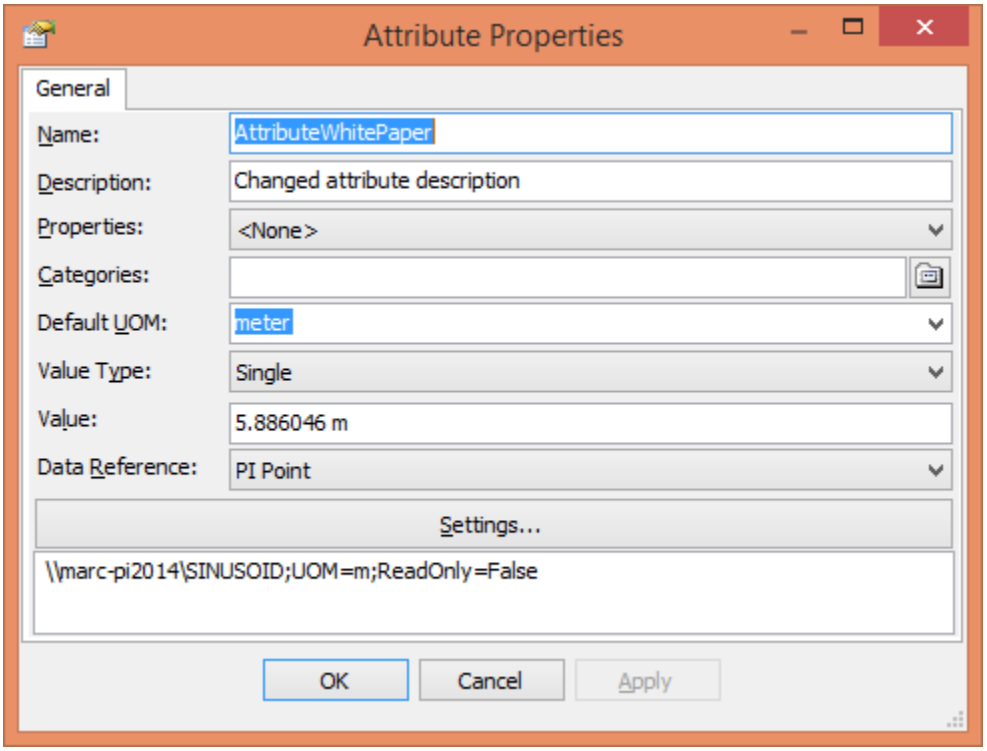


Figure 12– Changing the attribute description

Let's go back to Fiddler and change the cache header to "Cache-Control: max-age=600." This means that PI Web API will only update its AF Objects by making a call to the PI System if they were created more than 10 minutes before the current request execution. As this probably won't be true, if you click on "Execute", you will receive a JSON response with the old description.

Changing the cache header back to "Cache-Control: max-age=0" and executing this request again, you will receive a JSON with the updated attribute description as shown on Figure 13.

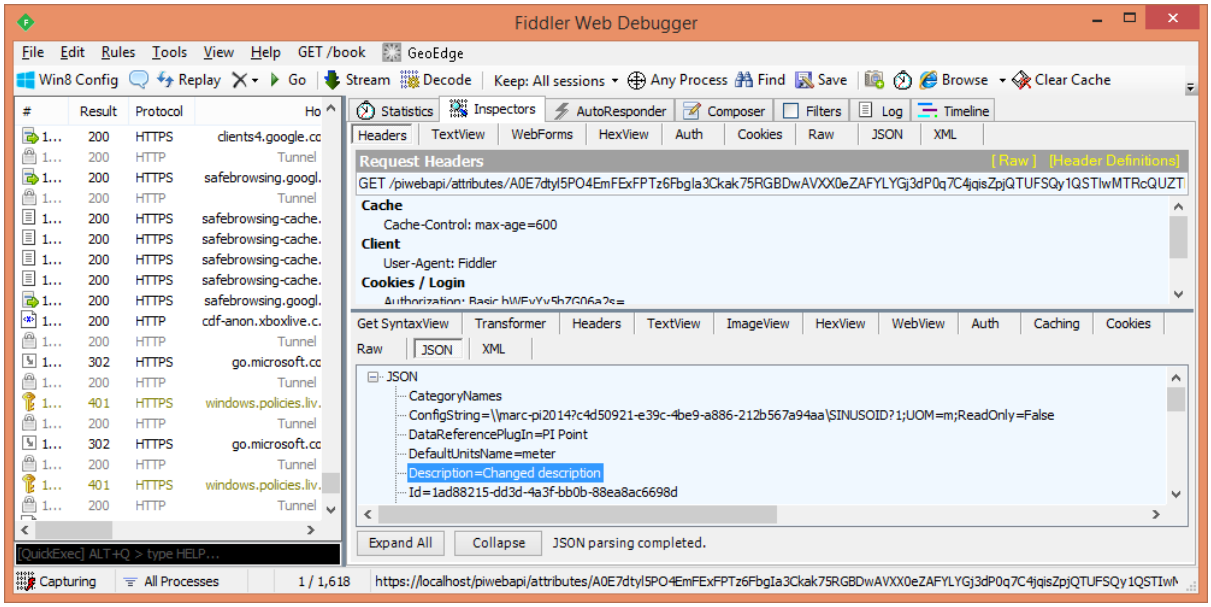


Figure 13– Getting the updated JSON response with Fiddler

We cannot recommend a value for the max-age parameter. You need to understand what your application needs and set up this value according to this context. The important part is to know the advantages and disadvantages from each option. If your application uses the cache, PI Web API will make fewer calls to the PI System and therefore, it will provide a faster response. On the other hand, using cache might not provide the most recent version of the AF Object as you have realized on this section.

3. Programming with PI Web API

One of the benefits on RESTful web services is that it enables cross-platform development of web, desktop, and mobile applications across many different programming languages. The purpose of this chapter is to help you get started developing with PI Web API on the following programming platforms:

- ASP.NET - C#
- JavaScript/jQuery
- PHP
- Java
- Matlab
- R

You can download the source code package stored on GitHub, which contains all the projects from the programming languages above. One of the prerequisites of running the examples provided on the following sections is to import the AF database and PI Points using files from the source code package.

For all different programming languages, we are going to create an application or a script that will update a value on a specific attribute. We are using an AF database with an element template called *Cities* with 6 attributes templates:

- Cloud Cover (%)
- Humidity (%)
- Pressure (mbar)
- Temperature (°C)
- Visibility (km)
- Wind Speed (%)

The XML file also contains five elements derived from this element template whose names are:

- Chicago
- Los Angeles
- New York
- San Francisco
- Washington

If you download the source code package, you will be able to import the XML file provided using PI System Explorer in order to build the sample AF database. On the source code

package, there is also a spreadsheet to create the necessary PI Points using PI Builder. The PI Points will show random values using the PI Random Interface. Once the database is created, you will have similar objects on PI System Explorer as shown on Figure 14:

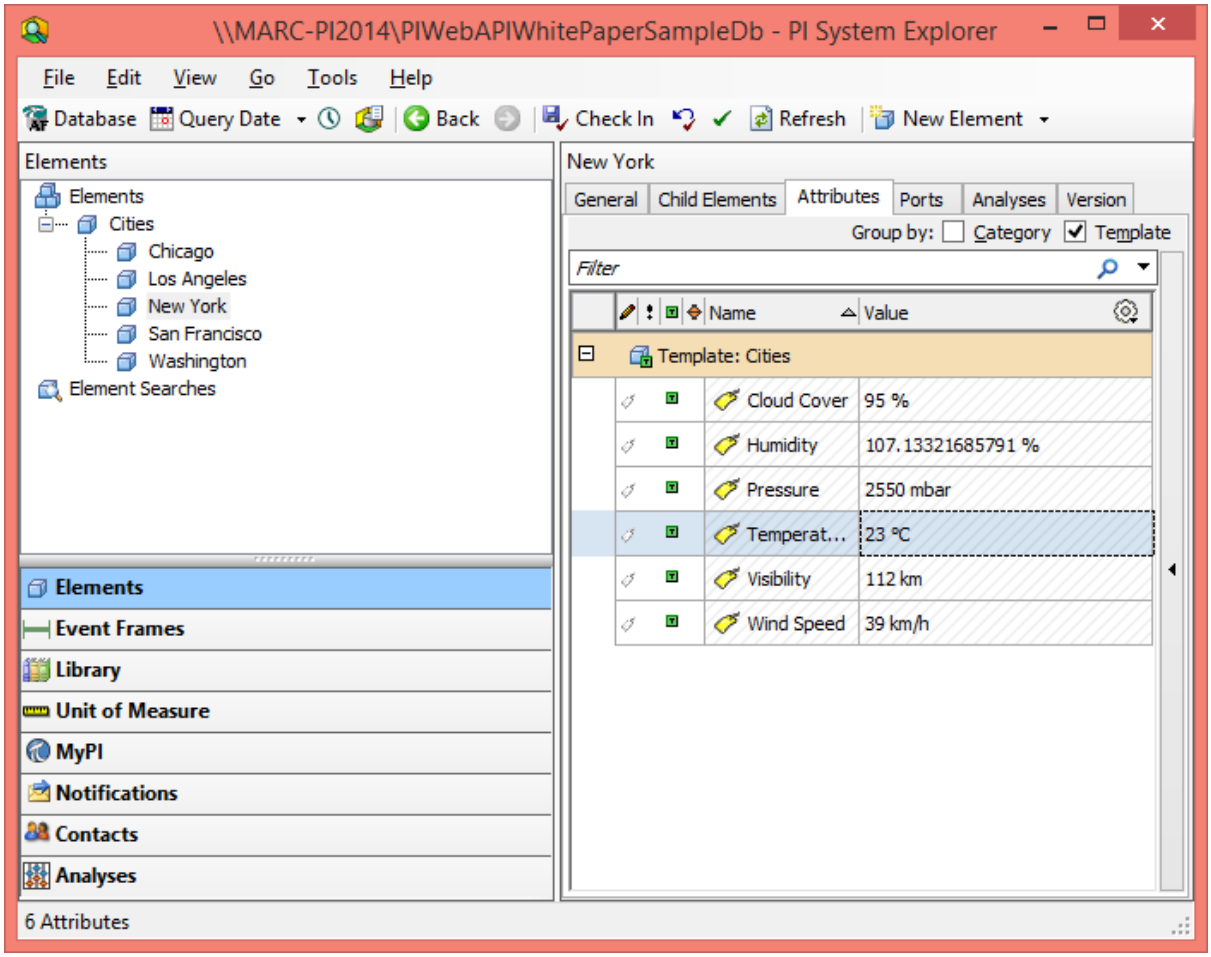


Figure 14 – AF database elements on PI System Explorer

For each programming language, we are going to develop an application or script that will:

1. Get the names from all cities stored on the AF database
2. Get the names from all attribute templates from the city template.
3. Choose a city and an attribute.
4. Update the attribute/PI Point by sending a new snapshot with a custom value through PI Web API.

Setting up PI Web API

In order for the six applications to work successfully, PI Web API needs to be properly configured. The configuration used are not be the same for all the projects. Kerberos and Basic authentication should be used on the ASP.NET MVC and JavaScript/jQuery examples. The other four projects would only work if the authentication attribute of PI Web API is set to anonymous. The reason is that the purpose of those projects is to get started developing with

PI Web API in many different languages. As it is more secure to work using Basic or Kerberos authentication, you might have to research about how to add an authentication header once you have selected a language to work with.

Remember that anonymous authentication with writes enabled is not a recommended configuration.

We have added hard-coded WebIDs on code snippets provided in our examples in order to make our samples simpler and more objective. Nevertheless, this is not a good practice. A better approach is to navigate the resource hierarchy dynamically using links or the get-by-path methods.

Please also make sure that:

- CorsHeader: "*"
- CorsOrigin: the domain of your web application (i.e http://localhost:54622)
- CorsMethods are set to: "GET,POST, PATCH, OPTIONS."
- CorsSupportsCredentials is set to True.
- DisableWrites is set to: False.

Please refer to the PI Web API Configuration section of the User Guide for information about how to edit the settings above.

You can find more information about using CORS in our PI Live Library.

ASP.NET MVC - C#

We are going to create an ASP.NET 5 MVC application in Visual Studio 2013. It is possible to do the same with Visual Studio 2010 or 2012 with ASP.NET MVC 4. Nevertheless, we have not tested on this version so small changes might be required in order to work properly. Please follow the steps below:

1. Create a new ASP.NET Web Application on Visual Studio.

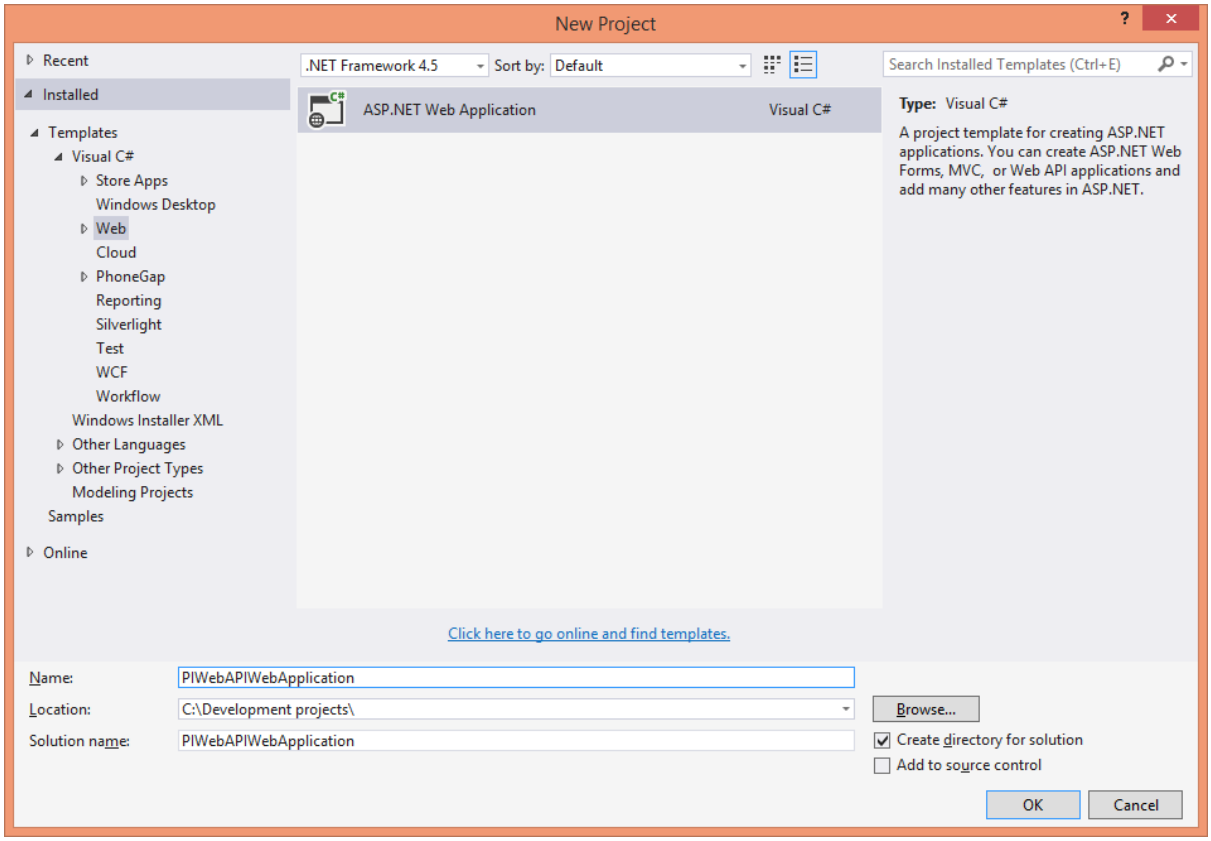


Figure 15 – Creating a new ASP.NET Web Application with Visual Studio

2. Choose the Empty template but add the folders and core references for MVC.

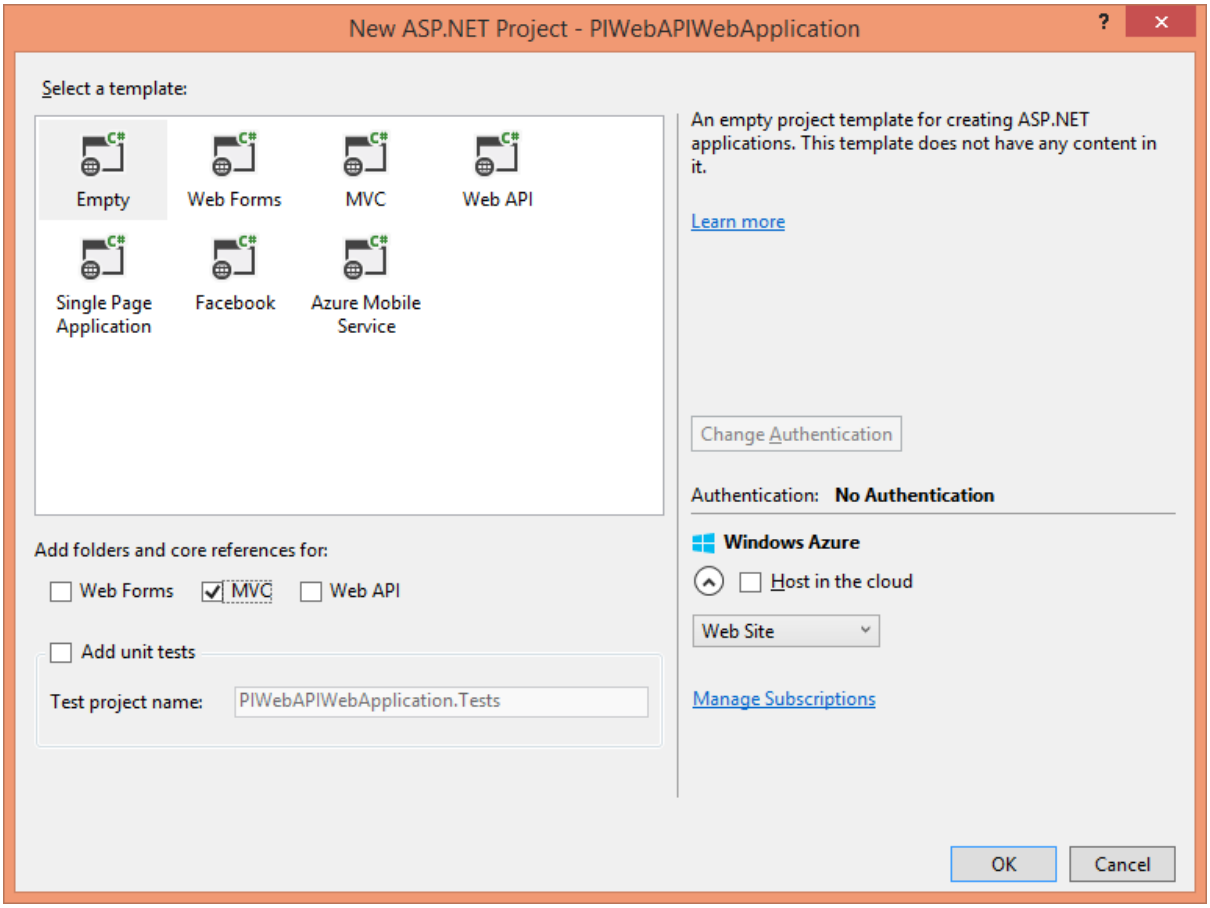


Figure 16 – Selecting the Empty template with the MVC core references

3. Go to Tools → NuGet Package Manager → Package Manager Console. Type "Install-Package Newtonsoft.json" to install JSON.NET, our .NET assembly to convert JSON responses to dynamic objects.
4. Create a new folder called "Content" and copy to this folder the default.css file provided with the supported files of this white paper. This file will only improve the design of the page and therefore, this step is not mandatory.
5. Create a new folder "Infrastructure" on the root folder and then the class file PIWebAPIWrapper.cs in it. The content of this file is below with some comments to understand how it works. Don't forget to get the WebIDs from your environment and update the WebIDs from the code snippet.

```
using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;
```

```
using System.Text;

using System.Web;

using System.Web.Mvc;


namespace PIWebAPIWebApplication.Infraestructure

{

    public class PIWebAPIWrapper

    {

        //Store the base url of PI Web API to be used on the other methods.

        private static string base_url = "https://marc-web-

sql.marc.net/piwebapi/";


        //This method will get all city names stored on PI AF.

        public static List<SelectListItem> GetCitiesName()

        {

            //SelectListItem is an ASP.NET MVC object used for the drop down list.

            List<SelectListItem> items = new List<SelectListItem>();


            //This url is shows the child elements from the Cities element. We

could have searched it by path

            //but we would have to make an extra REST call.

            string url = base_url +

"elements/E07dtyl5PO4EmFExFPTz6FbghNYw7Q65Uav8IhHRRw_WQTUFSQy1QSTIwMTRcQUZQQVJUTkV

SQ09VUlNFV0VBVEhFUlxDSVRJRVM/elements";

            dynamic result = MakeGetRequest(url);

            foreach (var item in result.Items)

            {

                items.Add(new SelectListItem { Text = item.Name.Value, Value =

item.Name.Value });

            }

            return items;

        }


        //This method will get all the attribute templates from the city template

stored on PI AF.

        public static List<SelectListItem> GetAttributesName()
```

```
        {

            List<SelectListItem> items = new List<SelectListItem>();

            string url = base_url +
"elementtemplates/T07dtyl5PO4EmFExFPTz6FbgSnYaLt6aAkqQMOBOaLJPKQTUFSQy1QSTIwMTRcQU
ZQQVJUTkVSQ09VUlNFV0VBVEhFUlxFTEVNRU5UVEVNUExBVEVTW0NJVElFUyBURU1QTEFURV0/attribut
etemplates";

            dynamic result = MakeGetRequest(url);

            foreach (var item in result.Items)

            {

                items.Add(new SelectListItem { Text = item.Name.Value, Value =
item.Name.Value });

            }

            return items;

        }



        //This method will update a value from a PI Point given a city and an
attribute.

        public static int SendValue(string cityName, string attributeName, string
value)

        {

            //We need to get the stream url of the attribute to be updated by
making a GET request as shown below.

            var url = base_url +

"attributes?path=\\\\MARC-PI2014\\AFPartnerCourseWeather\\Cities\\" + cityName +
"|" + attributeName;

            dynamic result = MakeGetRequest(url);

            //This will return the correct stream url to update the value.

            string url_to_send = result["Links"]["Value"];



            //Post Http request requires a requst body message. Please refer for
the PI Web API online help to know how it should be structured.

            string postData = "{'Value': " + value + " }";

            int statusCode = -1;

            MakePostRequest(url_to_send, postData, out statusCode);

            return statusCode;

        }
```

```
        //This function is responsable for the HTTP GET request

        internal static dynamic MakeGetRequest(string url)

        {

            WebRequest request = WebRequest.Create(url);


            //If you are not using Anonymous for PI Web API authentication, then
you need to provide your credentials.


            //Kerberos Authentication Only

            //request.UseDefaultCredentials = true;


            //Basic and Kerberos Authentication

            request.Credentials = new NetworkCredential("username", "password");


            WebResponse response = request.GetResponse();

            using (StreamReader sw = new
StreamReader(response.GetResponseStream()))

            {

                //After storing the JSON response on the StreamReader sw object,
we need to convert to a dynamic object.

                using (JsonTextReader reader = new JsonTextReader(sw))

                {

                    return JObject.ReadFrom(reader);

                }

            }

        }


        //This function is responsable for the HTTP POST request

        internal static void MakePostRequest(string url, string postData, out int
statusCode)

        {

            WebRequest request = WebRequest.Create(url);


            //Kerberos Authentication Only

            //request.UseDefaultCredentials = true;
```

```
            //Basic and Kerberos Authentication
            request.Credentials = new NetworkCredential("username", "password");

            ((HttpWebRequest)request).UserAgent = ".NET Framework Example Client";

            //The method is "GET" by default.
            request.Method = "POST";

            //If the content-type is not specified as application/json, your post
request won't work.
            request.ContentType = "application/json";

            //The content length should be present on the request header.
            byte[] byteArray = Encoding.UTF8.GetBytes(postData);
            request.ContentLength = byteArray.Length;
            Stream dataStream = request.GetRequestStream();
            dataStream.Write(byteArray, 0, byteArray.Length);
            dataStream.Close();

            //Get the response to make sure the value was sent.
            WebResponse response = request.GetResponse();

            //The StatusCode property is available only if the WebRequest object
is converted to HttpWebRequest
            //The StatusCode will confirm that the operation was executed
successfully.
            statusCode =
Convert.ToInt32(((System.Net.HttpWebResponse)(response)).StatusCode);
        }
    }
}
```

6. Remember to change the base_url to point to your PI Web API Server and also the two urls from the method GetCitiesName() and GetAttributesName().
7. Create a new class MainModel.cs under the Models with the following content:

```
using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

using System.Web.Mvc;


namespace PIWebAPIWebApplication.Models

{

    public class MainModel

    {

        public IEnumerable<SelectListItem> CitiesNames = null;

        public IEnumerable<SelectListItem> AttributesNames = null;

        public MainModel(IEnumerable<SelectListItem> downloadedCitiesNames,
IEnumerable<SelectListItem> downloadedAttributesNames)

        {

            CitiesNames = downloadedCitiesNames;

            AttributesNames = downloadedAttributesNames;

            Value = string.Empty;

        }

        public MainModel()

        {

            Value = string.Empty;

        }


        [Display(Name = "Select city:")]

        public string SelectedCity { get; set; }

        [Display(Name = "Select attribute:")]

        public string SelectedAttribute { get; set; }

        [Display(Name = "Value to send:")]

        public string Value { get; set; }

        [Display(Name = "Status:")]

        public string StatusMessage { get; set; }

    }
```

```
}
```

8. Right-click on the controller folder and choose Add → Controller… Select “MVC 5 Controller – Empty” and click on “Add.” On the “Add Controller” window, type HomeController on the Controller name field. Click on “Add” and edit the file with:

```
using PIWebAPIWebApplication.Infraestructure;

using PIWebAPIWebApplication.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;


namespace PIWebAPIWebApplication.Controllers

{

    public class HomeController : Controller

    {

        public ActionResult Index()

        {

            List<SelectListItem> citiesNames = PIWebAPIWrapper.GetCitiesName();

            List<SelectListItem> attributesNames =
PIWebAPIWrapper.GetAttributesName();

            MainModel mainModel = new MainModel(citiesNames, attributesNames);

            return View(mainModel);

        }


        [HttpPost]

        public ActionResult Index(MainModel mainModel)

        {


            mainModel.CitiesNames = PIWebAPIWrapper.GetCitiesName();

            mainModel.AttributesNames = PIWebAPIWrapper.GetAttributesName();
```

```
            int statusCode = PIWebAPIWrapper.SendValue(mainModel.SelectedCity,
mainModel.SelectedAttribute, mainModel.Value);

            if (statusCode == 202)

            {

                mainModel.StatusMessage = "Success!";

            }

            else

            {

                mainModel.StatusMessage = "Error";

            }

            return View("Index", mainModel);

        }

    }

}
```

9. Still on the HomeController.cs, right-click on the first View() method and choose "Add View…." Uncheck the option "Use a layout page:" and click on "Add." A new file Index.cshtml will be created under the Views\Home folder.
10. Edit the Index.cshtml with the content below:

```
@model PIWebAPIWebApplication.Models.MainModel

@{

    Layout = null;

    string showMessage = "true";

    if (string.IsNullOrEmpty(Model.StatusMessage))

    {

        showMessage = "false";

    }

}


<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <link href="~/Content/default.css" rel="stylesheet" />

    <script src="http://code.jquery.com/jquery-1.11.1.js"></script>

    <script src="~/Scripts/script.js"></script>
```

```
    <title>Sending Value to the PI System through PI Web API</title>
</head>
<body>
    @using (Html.BeginForm("Index", "Home", FormMethod.Post, null))
    {
        <h1>Sending Value to the PI System through PI Web API</h1>
        <p>Select the element and attribute that you want to send your value:</p>

        @Html.LabelFor(m => m.SelectedCity)
        @Html.DropDownListFor(m => m.SelectedCity, Model.CitiesNames)
        @Html.LabelFor(m => m.SelectedAttribute)
        @Html.DropDownListFor(m => m.SelectedAttribute, Model.AttributesNames)

        @Html.LabelFor(m => m.Value)
        @Html.TextBoxFor(m => m.Value)
        <br />
        <input type="submit" id="UpdateBtn" value="Send Value" />
    }

    <script>
        if (@showMessage) {
            alert('@Model.StatusMessage');
        }
    </script>
</body>
</html>
```

11. It is time to test. Click on Debug → Start Debugging. You will realize that the drop down list next to “Select city” will have their options filled with the elements on the AF attribute. The same happens with the attribute templates from the city template with the other drop down list. After selecting a city and an attribute, type a value on the text box and click on “Send Value.” If it works successfully, you will receive a success message.

Using WebRequest.UseDefaultCredentials = true works only when PI Web API is set up for Kerberos authentication. Basic authentication works by setting up the credentials property with an instance from the NetworkCredential class or by adding an authentication header.

The focus of this chapter is to help you use PI Web API on different platforms. This section has explained how to use PI Web API on C# by creating an ASP.NET MVC application. This is why we have not commented on some ASP.NET MVC techniques as it is out of scope from this white paper.

The class WebRequest was used in order to make PI Web API calls. Another alternative is to use HttpClient instead. If you are interested in this second approach, please refer to this blog post.

The screenshot below shows how the web page looks like on the browser. The ASP.NET MVC, JavaScript and PHP samples application looks very similar on the browser.

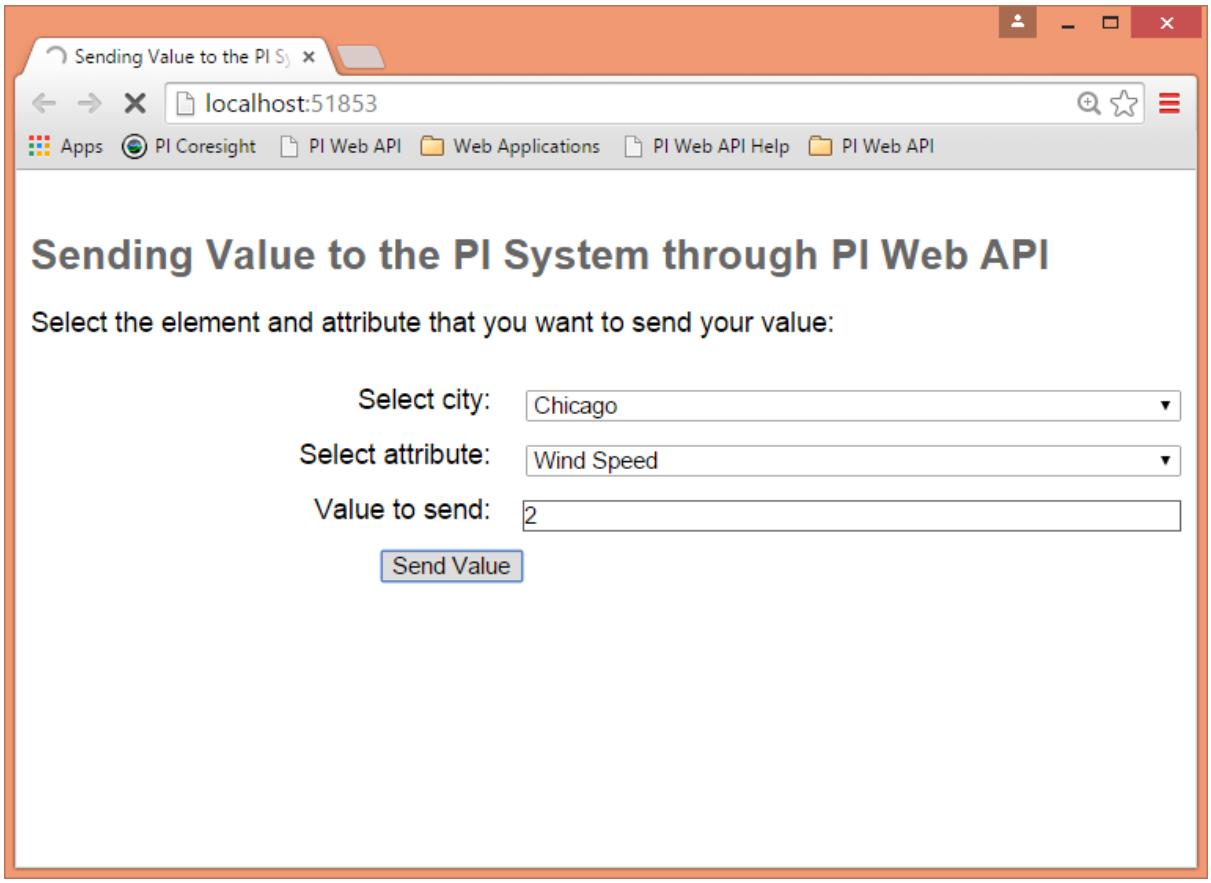


Figure 17 – Sample application web page for ASP.NET MVC, JavaScript and PHP

JavaScript

On this section, we are going to develop the same example using JavaScript and the jQuery library for the Ajax requests to PI Web API. The main difference between using JavaScript for the PI Web API calls or a server language such as ASP.NET MVC or PHP is where the calls

against PI Web API are made. On the previous example for ASP.NET, the web server was responsible for getting data from PI Web API which means that the end-user is not aware about the existence of PI Web API. If using JavaScript to do this task, each client browser will call PI Web API instead of the web server.

As the browser are communicating with PI Web API, CORS needs to be properly configured otherwise it will appear an error message.

We are going to use Google Chrome and Visual Studio to develop this sample application. You can use a text editor such as Notepad++ if you want to. But it is preferable to take advantage from all Visual Studio HTML features available.

1. Create a new ASP.NET Web Application the same way we did on the ASP.NET section.

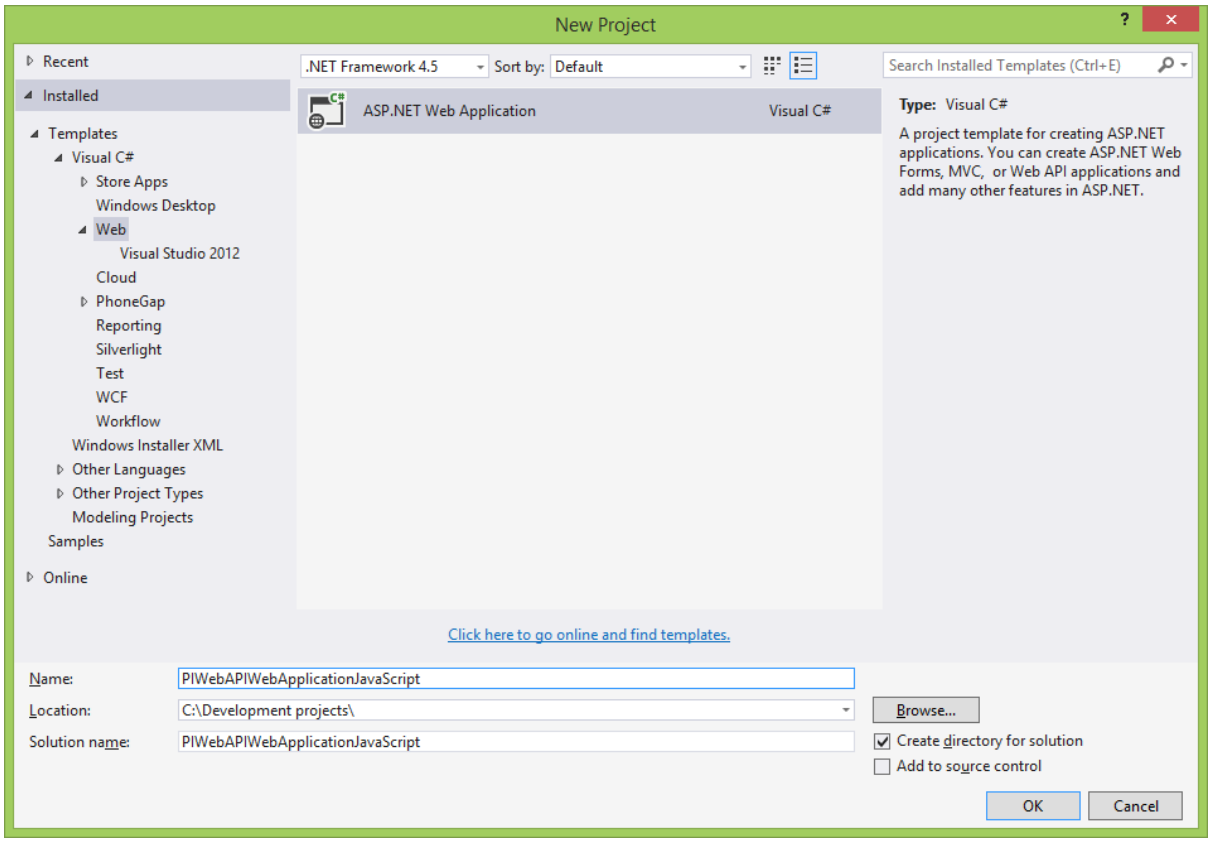


Figure 18 – Creating a new ASP.NET Web Application with Visual Studio

2. This time don't choose any folders and core references to be added to the project.

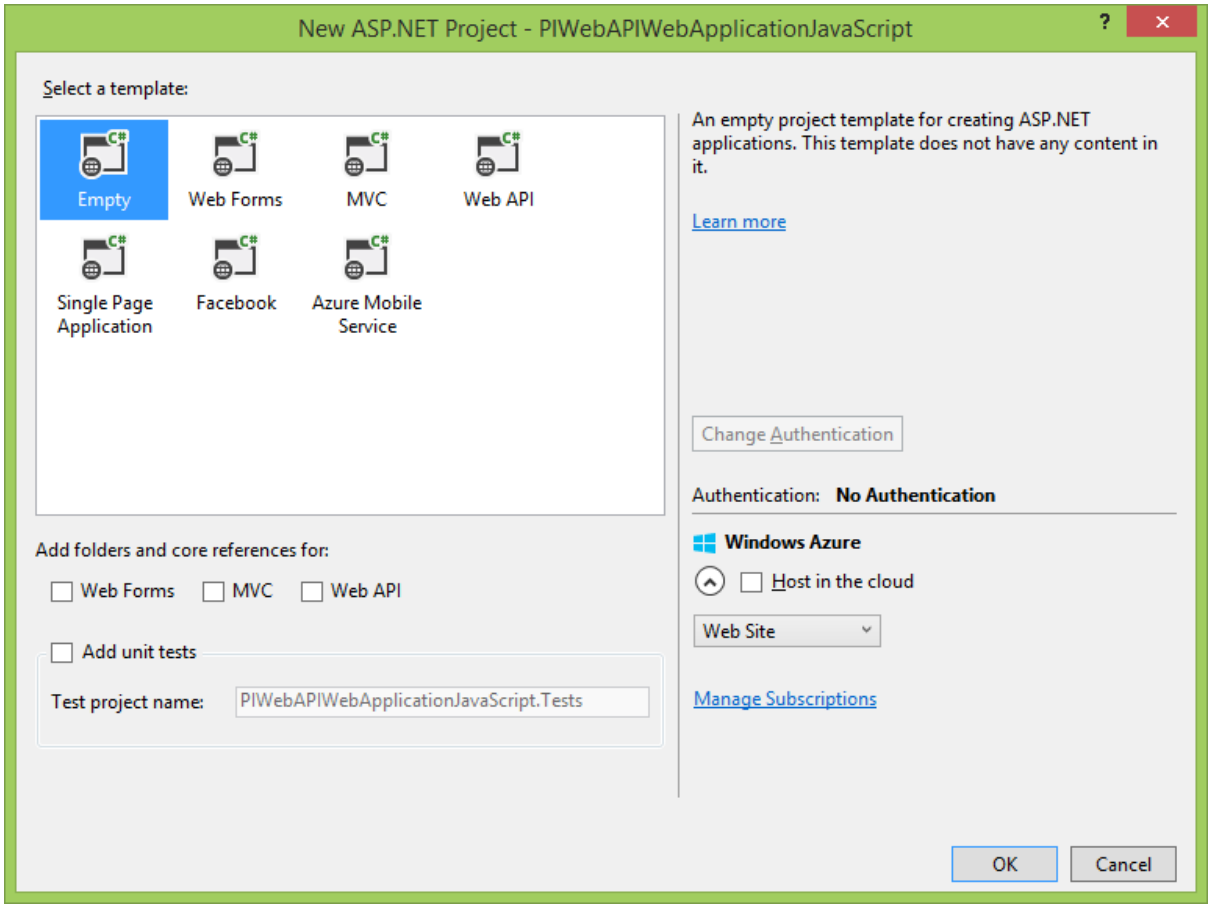


Figure 19 – Selecting the empty template with no core references

3. Create two folders on the root of the project: css and js. Add the default.css from the support files to the css folder and create a new file called script.js on the js folder. Finally, create the index.html file on the root.
4. The content of the index.html file would be:

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <link href="css/default.css" rel="stylesheet" />

    <script src="http://code.jquery.com/jquery-1.11.1.js"></script>

    <script src="js/script.js"></script>

    <title>Sending Value to the PI System through PI Web API</title>

</head>

<body>

    <h1>Sending Value to the PI System through PI Web API</h1>

    <p>Select the element and attribute that you want to send your value:</p>

    <label for="city_name">Select city</label>
```

```
    <select name="city_name" id="city_name" size="1"></select>

    <label for="attribute_name">Select attribute:</label>

    <select name="attribute_name" id="attribute_name" size="1"></select>

    <label for="value">Value to send:</label>

    <input type="text" name="value" id="value" value="" />

    <br />

    <input type="button" id="UpdateBtn" value="Send Value" onclick="SendValue()"
/>

</body>

</html>
```

5. As you can see, the structure of the HTML page is very similar to the previous example. The drop down lists are empty because they will be updated after receiving the json response from the PI Web API.
6. This approach provides a better user experience compared to previous example. The main reason is that on this case, the page first loads then it updates the drop down lists by making RESTful calls to PI Web API. On the ASP.NET MVC example, the controller would first get data from PI Web API then it would render the page. This means the ASP.NET MVC page shall take more time to load than the JavaScript version.
7. The script.js file has the content below. Don't forget to get the WebIDs from your environment and update the code snippet.

```
var base_url = "https://marc-web-sql.marc.net/piwebapi/";



//Function to get the cities names, which are child elements from the Cities
element.

function GetCitiesName() {

    var url = base_url +

"elements/E07dtyl5PO4EmFExFPTz6FbghNYw7Q65Uav8IhHRRw_WQTUFSQy1QSTIwMTRcQUZQQVJUTkV
SQ09VUlNFV0VBVEhFUlxDSVRJRVM/elements";

    MakeAjaxRequest('GET', url, function (data) {

        for (var i = 0; i < data.Items.length; i++) {

            $('#city_name').append($('<option>', {

                value: data.Items[i].Name,

                text: data.Items[i].Name
```

```
            }));

        }

    });

}

//Function to get the attributes template names, which are attribute templates
from the city element template.

function GetAttributesName() {

    var url = base_url +
"elementtemplates/T07dtyl5PO4EmFExFPTz6FbgSnYaLt6aAkqQMOBOaLJPKQTUFSQy1QSTIwMTRcQU
ZQQVJUTkVSQ09VUlNFV0VBVEhFUlxFTEVNRU5UVEVNUExBVEVTW0NJVElFUyBURU1QTEFURV0/attribut
etemplates";



    MakeAjaxRequest('GET', url, function (data) {

        for (var i = 0; i < data.Items.length; i++) {

            $('#attribute_name').append($('<option>', {

                value: data.Items[i].Name,

                text: data.Items[i].Name

            }), null);

        }

    });

}


//Function used to send a value to a PI Point/Attribute

function SendValue() {

    //Get the value of the text box with jQuery

    var value = $("#value")[0].value;


    //Get the city name with jQuery

    var cityName = $("#city_name")[0].value;


    //Get the attribute name with jQuery

    var attributeName = $("#attribute_name")[0].value;


    //Generate a url whose response will contain the other url to update the value

    var url = base_url +
```

```
"attributes?path=\\\\MARC-PI2014\\AFPartnerCourseWeather\\Cities\\" + cityName +
"|" + attributeName;

    var sendValueFunction = function (data) {

        var valueUrl = data["Links"]["Value"];

        MakeAjaxRequest('POST', valueUrl, function () {

            alert("Value has being sent successfully");

        }, "{'Value': " + value + " }");

    };


    //In this approach we are making a GET request. If it is successful, it will
make a POST request using the url from the GET response. The sendValueFunction
variable is actually a function.

    MakeAjaxRequest('GET', url, sendValueFunction, null);

}


function SendValue() {

    var value = $("#value")[0].value;

    var cityName = $("#city_name")[0].value;

    var attributeName = $("#attribute_name")[0].value;

    var url = base_url + "attributes?path=\\\\MARC-
PI2014\\AFPartnerCourseWeather\\Cities\\" + cityName + "|" + attributeName;

    var sendValueFunction = function (data) {

        var valueUrl = data["Links"]["Value"];

        MakeAjaxRequest('POST', valueUrl, function () {

            alert("Value has being sent successfully");

        }, "{'Value': " + value + " }");

    };

    //var sendValueFunction = function() {alert("OK!");}

    MakeAjaxRequest('GET', url, sendValueFunction, null);

}



function GetCitiesName() {

    var url = base_url +

"elements/E07dtyl5PO4EmFExFPTz6FbghNYw7Q65Uav8IhHRRw_WQTUFSQy1QSTIwMTRcQUZQQVJUTkV
SQ09VUlNFV0VBVEhFUlxDSVRJRVM/elements";
```

```
    MakeAjaxRequest('GET', url, function (data) {
        for (var i = 0; i < data.Items.length; i++) {
            $('#city_name').append($('<option>', {
                value: data.Items[i].Name,
                text: data.Items[i].Name
            }));
        }
    });
}

function GetAttributesName() {
    var url = base_url +
"elementtemplates/T07dtyl5PO4EmFExFPTz6FbgSnYaLt6aAkqQMOBOaLJPKQTUFSQy1QSTIwMTRcQU
ZQQVJUTkVSQ09VUlNFV0VBVEhFUlxFTEVNRU5UVEVNUExBVEVTW0NJVElFUyBURU1QTEFURV0/attribut
etemplates";

    MakeAjaxRequest('GET', url, function (data) {
        for (var i = 0; i < data.Items.length; i++) {
            $('#attribute_name').append($('<option>', {
                value: data.Items[i].Name,
                text: data.Items[i].Name
            }), null);
        }
    });
}

//This function calls the ajax method to make calls against PI Web API.
//It was possible to use the same function for GET and POST HTTP requests
//The difference is the type and the data.
//For Get requests, type='GET' and data is null;
//For Post requests, type='POST'and data is the request body.
function MakeAjaxRequest(type, url, SuccessCallBack, data) {
    $.ajax({
        type: type,
        url: url,
```

```
        cache: false,
        async: true,
        data: data,
        contentType: "application/json",
        username: 'useername',
        password: 'password',
        crossDomain: true,
        xhrFields: {
            withCredentials: true
        },
        success: SuccessCallBack,
        error: (function (error, variable) {
            console.log(error);
            alert('There was an error with the request');
        })
    });
}



$(document).ready(function () {
    //Once the page is loaded, both methods below will be executed.
    GetCitiesName();
    GetAttributesName();

});
```

8. This code worked successfully with Google Chrome using Basic and Kerberos authentication. There might be some issues with other browsers since not all of them behaves exactly the same. If you receive an error, this is probably related to:
    - AF attributes for PI Web API settings on the Configuration database, including CORS and authentication settings.
    - MakeAjaxRequest() method, which uses jQuery Ajax function internally, might need to be updated. For instance, if you are using Anonymous authentication, the options for username and password shall not be used.

If you don't provide the username and password as input parameters of the $.ajax constructor, the browser will open a window requesting the user credentials if PI Web API is set up with Basic or Kerberos authentication.

For more information about the ajax method, please refer to online documentation for jQuery. A blog post related to this topic was also published.

PHP DEVELOPMENT

The PHP application is also a server programming language as ASP.NET MVC. Therefore, the PHP application works the same way of the ASP.NET MVC application.

Make sure you have already set up an environment with Eclipse with Apache, PHP and XDebug.

I have already published two blog posts concerning PHP in case you are interested in:

- Introduction to PHP
- Developing a PHP application using PI Web API

The first article will teach you how to install PHP, set up Eclipse to let you program with PHP and configure XDebug for debugging. Reading the second article will help you develop a PHP application that would get data from the PI System and display on a web page through GET request. In this section we will continue developing the same application from this white paper and therefore it will show you how to use HTTP POST requests.

1. Open Eclipse and create a new PHP Application.

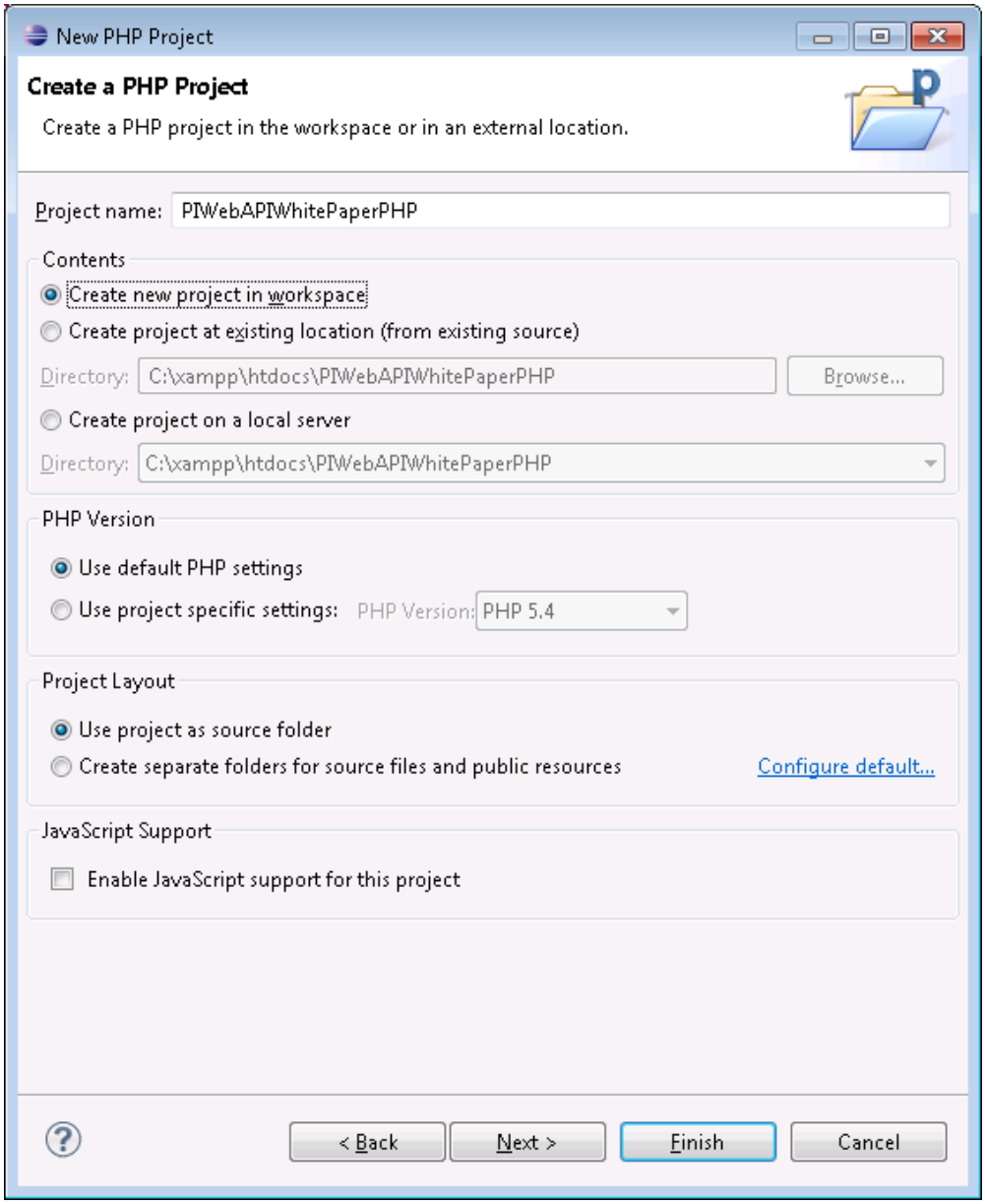


Figure 20 – Creating a new PHP project in Eclipse

2. Import the default.css from the supporting files packages to the root folder of the project.
3. Create two PHP files: index.php and piwebapi.class.php on the root folder of the project. Index.php is the file responsible for rendering HTML page while piwebapi.class.php works like the PIWebAPIWrapper.cs class from the ASP.NET MVC application. Let's start with the piwebapi.class.php. Don't forget to get the WebIDs from your environment and update the code snippet. The content of this file is:

```php
<?php
class PIWebAPIWrapper {
        public static function GetCitiesName() {
                $base_url = "https://osi-serv.osibr.com/piwebapi/";
                $url = $base_url .
"elements/E07dtyl5PO4EmFExFPTz6FbghNYw7Q65Uav8IhHRRw_WQTUFSQy1QSTIwMTRcQUZQQVJUTkV
SQ09VUlNFV0VBVEhFUlxDSVRJRVM/elements";
                $result = PIWebAPIWrapper::MakeGetRequest ( $url );
                $itemsList = array ();
                $i = 0;
                foreach ( $result->Items as $item ) {
                        $itemsList [$i] = $item->Name;
                        $i = $i + 1;
                }
                return $itemsList;
        }
        public static function GetAttributesName() {
                $base_url = "https://osi-serv.osibr.com/piwebapi/";
                $url = $base_url .
"elementtemplates/T07dtyl5PO4EmFExFPTz6FbgSnYaLt6aAkqQMOBOaLJPKQTUFSQy1QSTIwMTRcQU
ZQQVJUTkVSQ09VUlNFV0VBVEhFUlxFTEVNRU5UVEVNUExBVEVTW0NJVElFUyBURU1QTEFURV0/attribut
etemplates";
                $result = PIWebAPIWrapper::MakeGetRequest ( $url );
                $itemsList = array ();
                $i = 0;
                foreach ( $result->Items as $item ) {
                        $itemsList [$i] = $item->Name;
                        $i = $i + 1;
                }
                return $itemsList;
        }
        public static function SendValue($cityName, $attributeName, $value) {
                $base_url = "https://osi-serv.osibr.com/piwebapi/";
                $url = $base_url . "attributes?path=\\\\MARC-
PI2014\\AFPartnerCourseWeather\\Cities\\" . $cityName . "|" . $attributeName;
                $result = PIWebAPIWrapper::MakeGetRequest ( $url );
                $url_to_send = $result->Links->Value;
                $postData = "{'Value': " . $value . " }";
                $statusCode = PIWebAPIWrapper::MakePostRequest ( $url_to_send,
$postData );
                return $statusCode;
        }
        private static function MakeGetRequest($url) {
                $ch = curl_init ( $url );
                curl_setopt ( $ch, CURLOPT_HEADER, false );
                curl_setopt ( $ch, CURLOPT_RETURNTRANSFER, true );
                curl_setopt ( $ch, CURLOPT_SSL_VERIFYPEER, false );
                // header('Content-type: application/json');
                $result = curl_exec ( $ch );
                $json_o = json_decode ( $result );
                return ($json_o);
        }
        private static function MakePostRequest($url, $postData) {
                $ch = curl_init ( $url );
                curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST");
                curl_setopt ( $ch, CURLOPT_POSTFIELDS, $postData );
                curl_setopt ( $ch, CURLOPT_RETURNTRANSFER, true);
                curl_setopt ( $ch, CURLOPT_SSL_VERIFYPEER, false );
                curl_setopt($ch, CURLOPT_HTTPHEADER, array(
                'Content-Type: application/json',
```

```
                'Content-Length: ' . strlen($postData))
                );

                $result = curl_exec ( $ch );
                $status = curl_getinfo($ch, CURLINFO_HTTP_CODE);

                curl_close($ch);
                return (int) $status;
        }
}
?>
```

4. This works very similar to the C# and JavaScript but it is written on PHP. This code only works for the anonymous authentication option on PI Web API. Nevertheless, the function curl_setopt provides options to implement better security. Please refer to this page on php.net for more information on this topic. The content of index.html is:

```
<?php
    include_once ("piwebapi.class.php");
    $citiesNames = PIWebAPIWrapper::GetCitiesName ();
    $attributesNames = PIWebAPIWrapper::GetAttributesName ();
    $postAction = 0;
    if (isset ( $_POST ["value"] )) {
        $valueToSend = $_POST ["value"];
        $selectedCity = urlencode ($_POST ["city_name"]);
        $selectedAttribute = urlencode ($_POST ["attribute_name"]);
        $statusCode = PIWebAPIWrapper::SendValue ( $selectedCity,
$selectedAttribute, $valueToSend);
        $postAction = 1;
    }

?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <link href="default.css" rel="stylesheet" />
        <title>Sending Value to the PI System through PI Web API</title>
    </head>
    <body>
        <h1>Sending Value to the PI System through PI Web API</h1>
        <form action="index.php" method="post">
            <p>Select the element and attribute that you want to send your
value:</p>
            <label for="city_name">Select city</label>
            <select name="city_name" id="city_name" size="1">
                <?php
                    foreach ( $citiesNames as $cityName ) {
                        echo "<option value=\"$cityName\">" . $cityName .
"</option>";
                    }
                ?>

            </select> <label for="attribute_name">Select attribute:</label>
            <select name="attribute_name" id="attribute_name" size="1">
                <?php
                    foreach ( $attributesNames as $attributeName ) {
```

```
                        echo "<option value=\"$attributeName\">" . $attributeName
. "</option>";
                    }
                ?>
            </select> <label for="value">Value to send:</label> <input type="text"
name="value" id="value" value="" /> <br /> <input type="submit" id="UpdateBtn"
value="Send Value" />
        </form>
        <?php
            if ($postAction == 1) {
                echo "<script>";
                if ($statusCode==202)
                {
                    echo "alert('Success')";
                }
                else
                {
                    echo "alert('Error')";
                }
                echo "</script>";
            }
        ?>
    </body>
</html>
```

This index.php page first includes the piwebapi.class.php file and checks if the page is loaded for the first time or if it has received values from its form.

If it is loaded for the first time, it would get the cities names and attributes names and display the form. After those drop down lists are selected, the value to be sent is typed and the button “Send” is pressed, the page is reloaded but this time, PHP would detect that the form has sent data. Then, it tries to send the value, display the form again and display a success message in case the value was sent successfully by adding some JavaScript code.

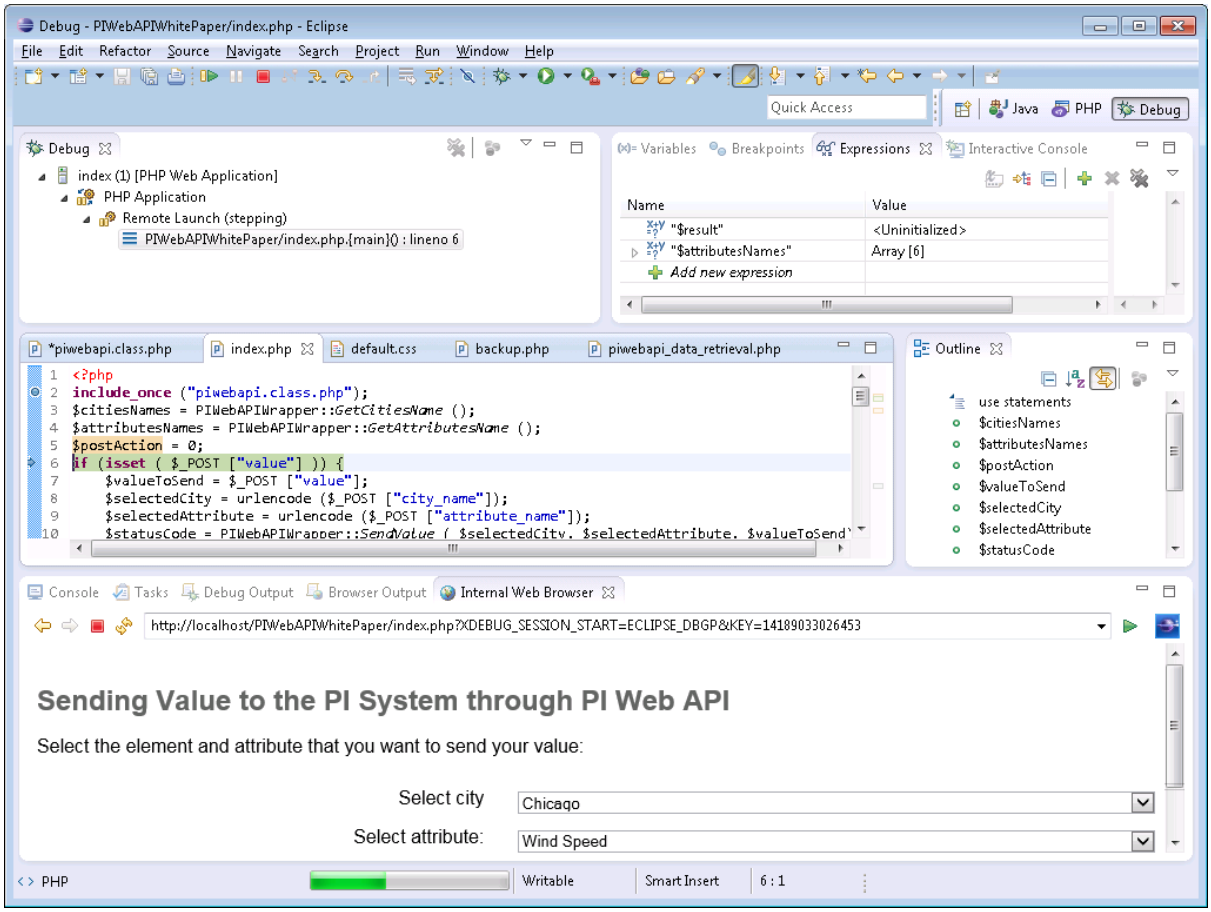


Figure 21 – Running the PHP application on Eclipse.

Java Development

For Java, Matlab and R examples, we are not going to develop a web application but a console application or script without UI.

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.

PI Web API is the new option for Java developers besides PI JDBC. On this example, we will be using Eclipse for Java Programing.

1. Before creating a new project, you need to download:
   a. Apache HttpComponents project libraries → this project is responsible for creating and maintaining a toolset of low level Java components focused on HTTP and associated protocols. Click here to go to their web site and download their libraries.
   b. Json simple → it is a simple Java toolkit for JSON. You can use JSON.simple to encode or decode JSON text. Click here to download.
   c. If you prefer you can get the libraries from the source code package.

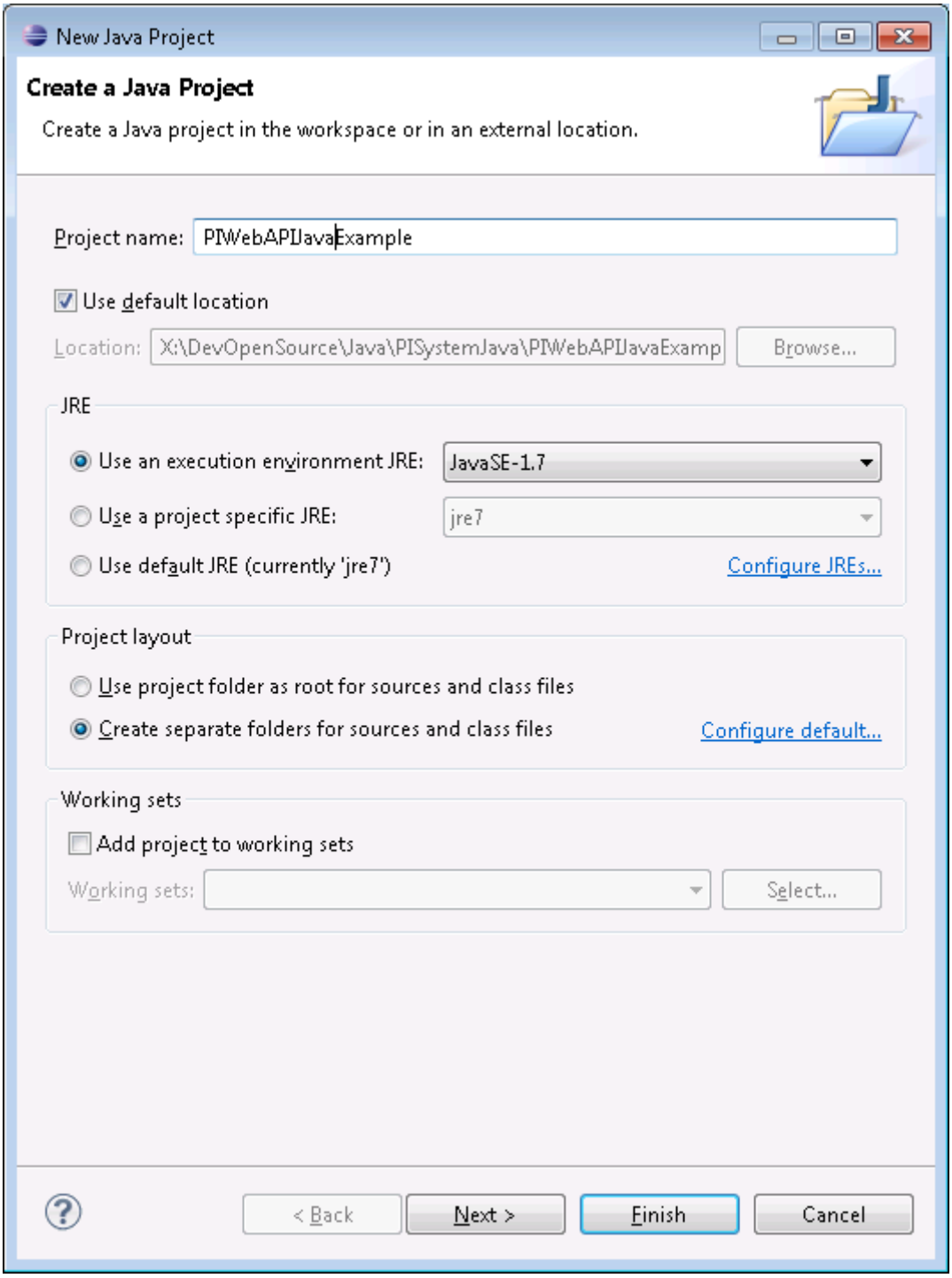


Figure 22 – Creating a Java project on Eclipse.

2. Create a lib folder under the root folder of the project. Copy all the jar files required to that folder.

3. We need to reference the jar libraries to the project. Right-click on the project and choose “Properties.” Under “Java Build Path”, and the “Libraries” ribbon, click on “Add JARs...” Select all the JAR files located on the lib folder and click on “OK.”
4. By this time, you should be seeing a similar screenshot:

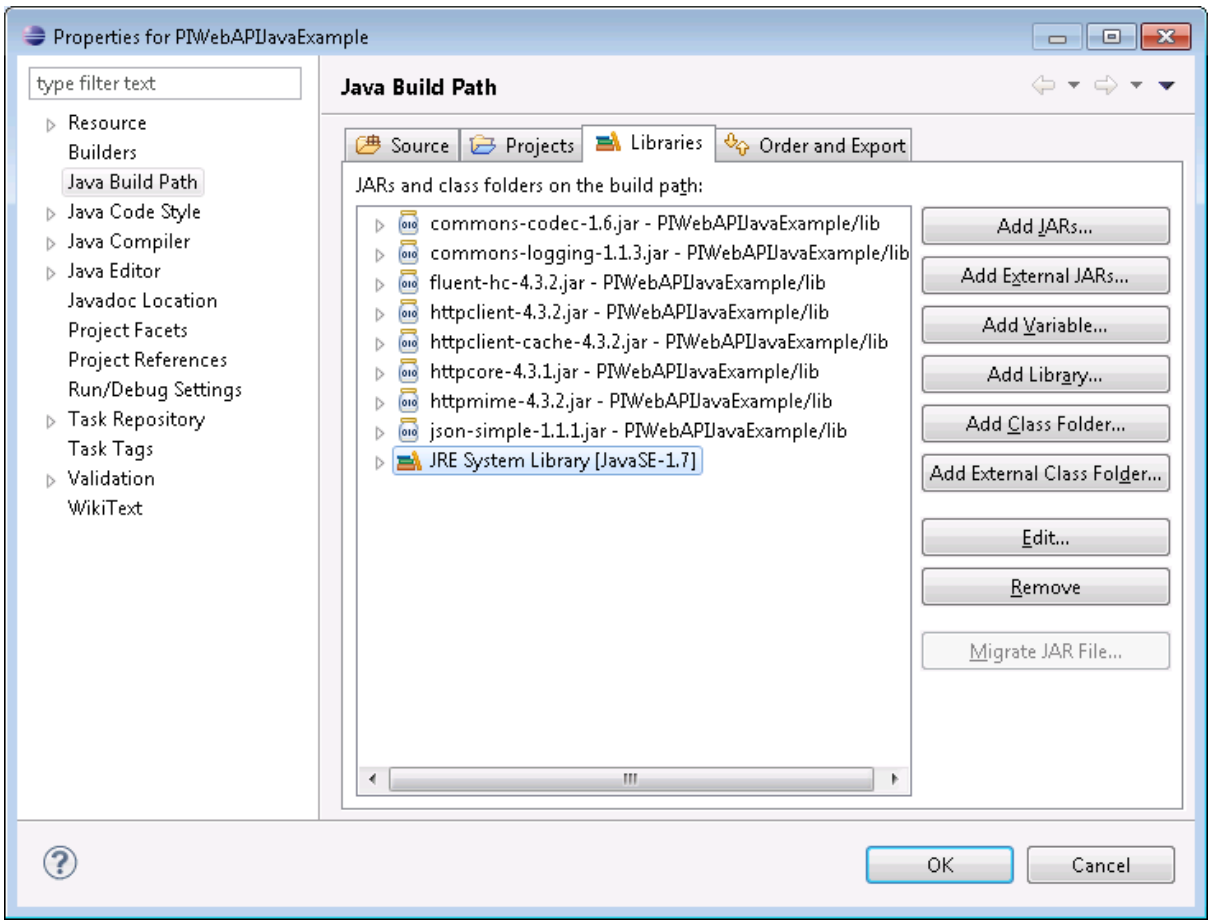


Figure 23 – Referencing JAR libraries on the Java project.

5. Create a new class with default package called PIWebAPIWrapper whose content should be:

```
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.*;
import org.apache.http.client.*;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
```

```
public class PIWebAPIWrapper {
    private static String base_url = "https://osi-
serv.osibr.com/piwebapi/";

    public static List<String> GetCitiesName() {
        List<String> items = new ArrayList<String>();
        String url = base_url                    +
"elements/E07dtyl5PO4EmFExFPTz6FbghNYw7Q65Uav8IhHRRw_WQTUFSQy1QSTIwMTRcQUZQ
QVJUTkVSQ09VUlNFV0VBVEhFUlxDSVRJRVM/elements";
        JSONArray citiesItems = (JSONArray)
MakeGetRequest(url).get("Items");
        for (int i = 0; i < citiesItems.size(); i++) {
            JSONObject myCity = (JSONObject) citiesItems.get(i);
            items.add(myCity.get("Name").toString());
        }
        return items;
    }

    public static List<String> GetAttributesName() {
        List<String> items = new ArrayList<String>();
        String url = base_url
                +
"elementtemplates/T07dtyl5PO4EmFExFPTz6FbgSnYaLt6aAkqQMOBOaLJPKQTUFSQy1QSTI
wMTRcQUZQQVJUTkVSQ09VUlNFV0VBVEhFUlxFTEVNRU5UVEVNUExBVEVTW0NJVElFUyBURU1QTE
FURV0/attributetemplates";
        JSONArray citiesItems = (JSONArray)
MakeGetRequest(url).get("Items");
        for (int i = 0; i < citiesItems.size(); i++) {
            JSONObject myCity = (JSONObject) citiesItems.get(i);
            items.add(myCity.get("Name").toString());
        }
        return items;
    }

    public static int SendValue(String cityName, String attributeName,
            String value) {
        String attributePath = "\\\\MARC-
PI2014\\AFPartnerCourseWeather\\Cities\\"
                + cityName + "|" + attributeName;
        String url = null;
        try {
            url = base_url + "attributes?path="
                    + URLEncoder.encode(attributePath, "UTF-8");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        JSONObject linkItems = (JSONObject)
MakeGetRequest(url).get("Links");
        String url_to_send = linkItems.get("Value").toString();
        try {
            return MakePostRequest(url_to_send, value);
        } catch (IOException e) {

            e.printStackTrace();
            return -1;
        }
    }


    private static JSONObject MakeGetRequest(String url) {
        JSONObject myJSONObject = null;
        try {
```

```
            HttpClient client = HttpClientBuilder.create().build();
            HttpGet request = new HttpGet(url);
            HttpResponse response = client.execute(request);
            HttpEntity entity = response.getEntity();
            String responseString = EntityUtils.toString(entity);

            JSONParser myJSONParser = new JSONParser();
            myJSONObject = (JSONObject) myJSONParser.parse(responseString);

        } catch (Exception ex) {

        }
        return myJSONObject;
    }

    private static int MakePostRequest(String url, String value)
            throws ClientProtocolException, IOException {
        HttpClient client = HttpClientBuilder.create().build();
        HttpPost post = new HttpPost(url);
        post.setHeader("Content-Type", "application/json");
        post.setEntity(new StringEntity("{\"Value\":" + value + "}"));
        HttpResponse response = client.execute(post);
        int responseCode = response.getStatusLine().getStatusCode();
        return responseCode;
    }
}
```

6. Create the second class called Application also with the default package. Change its content to:

```
import java.util.List;

public class Application {
    public static void main(String[] args)
    {
    List<String> citiesNames = PIWebAPIWrapper.GetCitiesName();
    List<String> attributesNames = PIWebAPIWrapper.GetAttributesName();
    String value = "1320";
    int statusCode = PIWebAPIWrapper.SendValue(citiesNames.get(0),
attributesNames.get(1), value);

        if (statusCode == 202)
        {
            System.out.println("Success!");
        }
        else
        {
            System.out.println("Error");
        }
    }
}
```

7. Run the Application.java and make sure you see the “Success!” output under the Console tab.

The PIWebAPIWrapper is very similar to the other programming language but written in Java with different libraries. Comparing with the ASP.NET MVC application, json-simple library replaces JSON.NET and the assembly HttpClient replaces WebRequest.

This example actually works as a script. Please take a look at the Application.java file. It will get an array of cities and another array for the attributes. Then, it will send the value 1320 to the second attribute from the attributes array associated with the first element of the cities array. If the status code received is equal to 202 then it will display the “Success!” message otherwise an error message.

Using keytool to import a certificate

If PI Web API is using a self-signed certificate, you need to set up your client to trust this certificate in Java through the keytool application. If not, you should receive the following exception when running this application:

javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target

1. In your browser, go to the HTTPS URL that Java could not access. Click on the HTTPS certificate chain (there is lock icon in the Internet Explorer and Google Chrome), click on the lock to view the certificate, as shown below:

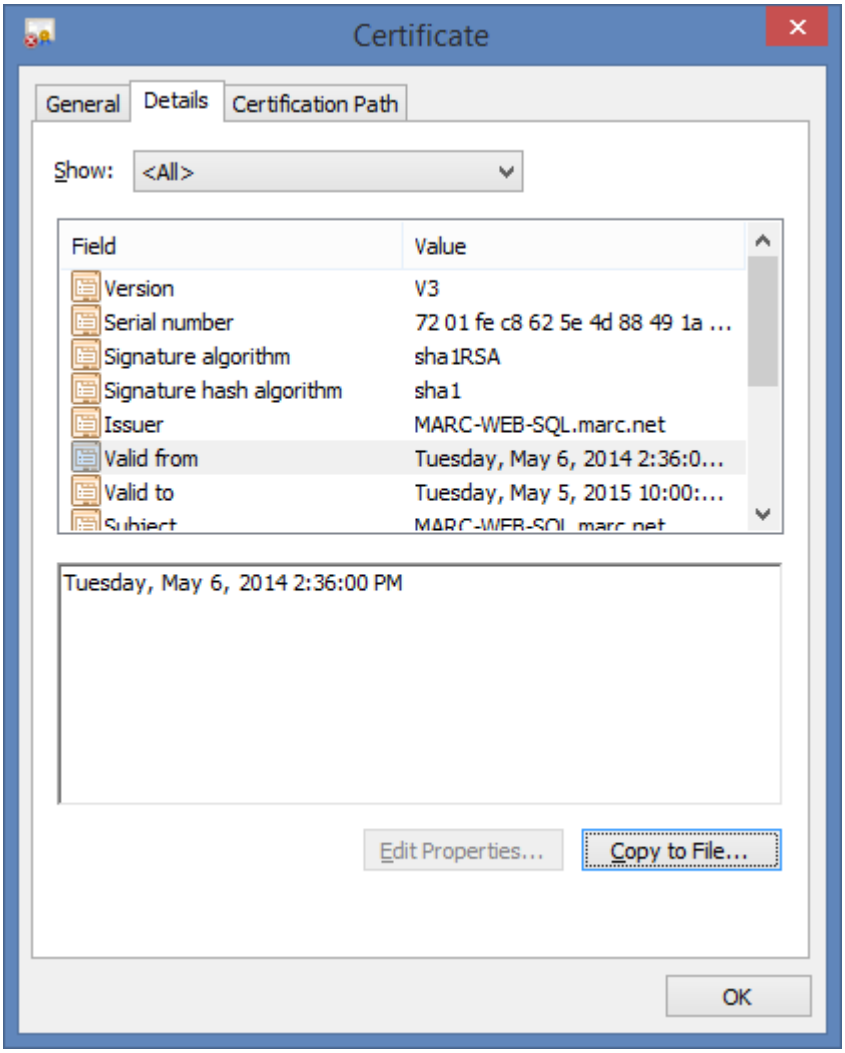


Figure 24 – Viewing the SSL certificate.

2. Go to “Details” of the certificate and click on “Copy to file.” Copy it in Base64 (.cer) format. It will be saved on your Desktop. Install the certificate ignoring all the alerts.

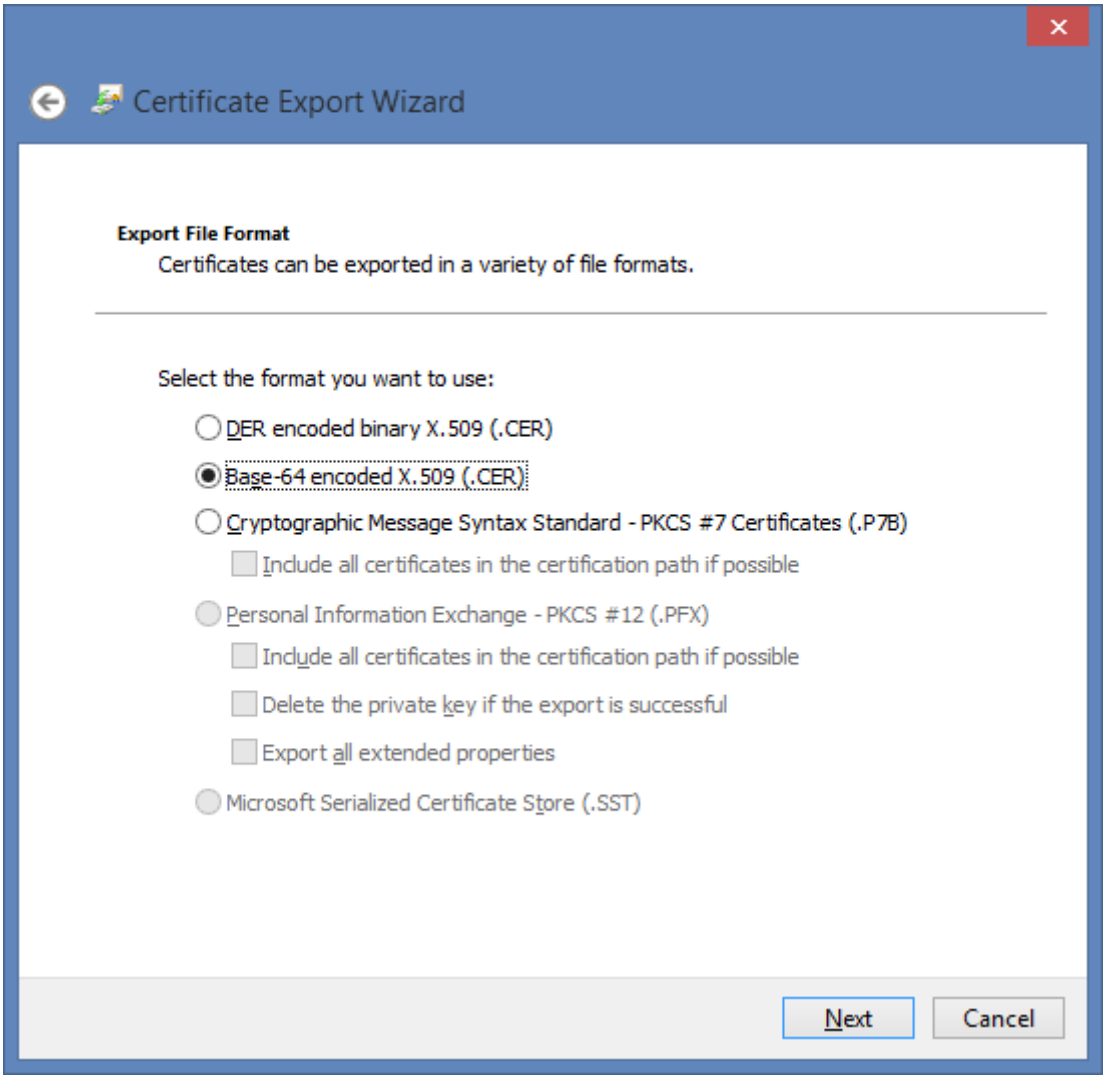


Figure 25 – Exporting the certificate to the .CER format.

3. Run keytool to import the certificate with the following command:

keytool -import -alias <new_unique_alias> -file <any_filename_here_from_above> -keystore <path/to/truststore>

In our case, we have used:

"C:\Program Files\Java\jre7\bin\keytool" -importcert -trustcacerts -alias PIWEBAPICERT -file c:\certificate.cer -keystore "C:\Program Files\Java\jre7\lib\security\cacerts" -storepass changeit

4. Close Eclipse and all the Java applications then reopen them and try again. If you have followed the steps properly, you won't receive that error message anymore.

MATLAB DEVELOPMENT

MATLAB is a multi-paradigm numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, Fortran and Python.

We will develop a similar script that would do similar tasks from the Java example.

To facilitate working with the JSON object in MATLAB, the following examples use JSONlab to parse JSON texts into MATLAB structures. JSONlab is available at http://iso2mesh.sourceforge.net/cgi-bin/index.cgi?jsonlab or http://www.mathworks.com/matlabcentral/fileexchange/33381-jsonlab--a-toolbox-to-encode-decode-json-files-in-matlab-octave. However, feel free to parse the JSON strings using other methods if you prefer.

1. The project has two files: PIWebAPIWrapper.m and Application.m. The content of the first file is:

```
classdef PIWebAPIWrapper < handle

   methods(Static)

      function citiesName = GetCitiesName()

        base_url = 'https://osi-serv.osibr.com/piwebapi/';

        url
=strcat(base_url,'elements/E07dtyl5PO4EmFExFPTz6FbghNYw7Q65Uav8IhHRRw_WQ
TUFSQy1QSTIwMTRcQUZQQVJUTkVSQ09VUlNFV0VBVEhFUlxDSVRJRVM/elements');

        result = PIWebAPIWrapper.MakeGetRequest(url);

        [~,y] = size(result.Items);

        citiesName = cell(1,y);

            for i=1:y

                citiesName{i} = result.Items{i}.Name;

            end

      end


     function attributesName = GetAttributesName()

        base_url = 'https://osi-serv.osibr.com/piwebapi/';

        url
=strcat(base_url,'elementtemplates/T07dtyl5PO4EmFExFPTz6FbgSnYaLt6aAkqQM
```

```
OBOaLJPKQTUFSQy1QSTIwMTRcQUZQQVJUTkVSQ09VUlNFV0VBVEhFUlxFTEVNRU5UVEVNUEx
BVEVTW0NJVElFUyBURU1QTEFURV0/attributetemplates');

        result = PIWebAPIWrapper.MakeGetRequest(url);

        [~,y] = size(result.Items);

        attributesName = cell(1,y);

            for i=1:y

                attributesName{i} = result.Items{i}.Name;

            end

    end


    function statusCode = SendValue(cityName, attributeName, value)

        base_url = 'https://osi-serv.osibr.com/piwebapi/';

        url =strcat(base_url,'attributes?path=\\MARC-
PI2014\AFPartnerCourseWeather\Cities\',cityName,'|',attributeName);

        result = PIWebAPIWrapper.MakeGetRequest(url);

        url_to_send = result.Links.Value;

        postData = strcat('{''Value'': ',value,' }');

        NET.addAssembly('System.Net');

        import System.Net.*

        import System.Text.*

        webrequest = WebRequest.Create(url_to_send);

        webrequest.Method = 'POST';

        webrequest.ContentType = 'application/json';

        byteArray = Encoding.UTF8.GetBytes(postData);

        webrequest.ContentLength = byteArray.Length;

        dataStream = webrequest.GetRequestStream();

        dataStream.Write(byteArray, 0, byteArray.Length);

        dataStream.Close();

        webresponse = webrequest.GetResponse();

        if (webresponse.StatusDescription=='Accepted')

           statusCode=202;

        else

           statusCode=404;

        end
```

```
    end


      function json_response = MakeGetRequest(url)
            strResponse = urlread(url);
            json_response = loadjson(strResponse);
      end
  end
end
```

2. The interesting thing of this example is that although the loadjson function (from JSONlab) was used together with the native Matlab function urlread to make the HTTP GET request against PI Web API, .NET Framework libraries are used for making the HTTP POST requests. Being able to develop in Matlab with .NET Framework libraries is a very interesting feature. It is possible for instance to use PI AF SDK within Matlab in case you are interested in. Please refer to our white paper "Using PI Data with Matlab" for more information about that topic.
3. The code for the Application.m file is:

```
clear all;

close all;

clc;

addpath('C:\Program Files\MATLAB\jsonlab');

citiesName =PIWebAPIWrapper.GetCitiesName();

attributesName = PIWebAPIWrapper.GetAttributesName();

value=50;


statusCode = PIWebAPIWrapper.SendValue(citiesName{1}, attributesName{2},
value);

if (statusCode==202)

    display('Success!');

else

    display('Error!');

end
```

4. In order to use JSONLab, we will first add the MATLAB toolbox JSONlab to the MATLAB search path:

```
addpath('C:\Program Files\MATLAB\jsonlab');
```

The other part of the code is very similar to the Java application version.

Note that if you are using a development environment and does not have a valid SSL certificate for PI Web API, you might get an error using MATLAB's urlread function. You will need to first import the untrusted certificate into the MATLAB's JRE keystore by following the procedures from http://www.mathworks.com/matlabcentral/answers/92506-can-i-force-urlread-and-other-matlab-functions-which-access-internet-websites-to-open-secure-websi. Alternatively, you can import the certificate manually by using keytool as it was shown on the previous section.

R Development

We will continue to develop our application/script in R. This R script would be able to exchange data with PI Web API directly. Nevertheless, keep in mind that there are other solutions to integrate PI Web API with R as well:

- Develop a Windows Form Application exchanging data with PI Web API through Web Request class and with R through R.NET.

- Develop a Java application with JRE libraries in order to show R graphics, also exchanging data with PI Web API.

There is a white paper about this topic named "Integrating the PI System with R." It provides several types of solutions including R.NET and JRE. It is important to study the different scenarios and options available in order to decide which is the most feasible solution that attends your requirements.

1. Back to our R script development, we need to install two libraries through R console. Please use the following commands to do so:
   a. install.packages("RCurl",lib=.Library)
   b. install.packages("rjson",lib=.Library)
2. Two files should be created: piwebapi.r and main.r. Let's start with the first one:

```
GetCitiesName<- function() {

base_url = 'https://osi-serv.osibr.com/piwebapi/';

url=paste(base_url,
'elements/E07dtyl5PO4EmFExFPTz6FbghNYw7Q65Uav8IhHRRw_WQTUFSQy1QSTIwMTRcQ
UZQQVJUTkVSQ09VUlNFV0VBVEhFUlxDSVRJRVM/elements',sep="");

result = MakeGetRequest(url);

l = length(result$Items);

citiesname <- 1:l
```

```
    for(i in 1:l){

     citiesname[i] =  result$Items[[i]]$Name;

    }

    x=citiesname;

}


GetAttributesName<- function() {

base_url = 'https://osi-serv.osibr.com/piwebapi/';

url
=paste(base_url,'elementtemplates/T07dtyl5PO4EmFExFPTz6FbgSnYaLt6aAkqQMO
BOaLJPKQTUFSQy1QSTIwMTRcQUZQQVJUTkVSQ09VUlNFV0VBVEhFUlxFTEVNRU5UVEVNUExB
VEVTW0NJVElFUyBURU1QTEFURV0/attributetemplates',sep="");

result = MakeGetRequest(url);

l = length(result$Items);

attributesname <- 1:l

    for(i in 1:l){

     attributesname[i] =  result$Items[[i]]$Name;

    }

    x=attributesname;

}


SendValue<- function(cityName, attributeName, value){

base_url = 'https://osi-serv.osibr.com/piwebapi/';

stringarray <- c(base_url, "attributes?path=\\\\MARC-
PI2014\\AFPartnerCourseWeather\\Cities\\",cityName,"|",attributeName)

url =paste(stringarray,collapse="");

result = MakeGetRequest(url);

url_to_send = result$Links$Value;

stringarray <- c('{\'Value\': ',value,' }')

postData = paste(stringarray,collapse="");

MakePostRequest(url_to_send,postData);

}


MakeGetRequest <- function(url) {

w=getURL(url,ssl.verifypeer = FALSE);
```

```
x=fromJSON(w);

}



MakePostRequest <- function(url, postData) {

headers <- list('Content-Type' = 'application/json')

rs = postForm(url, .opts=list(postfields=postData,
httpheader=headers,ssl.verifypeer = FALSE))

}
```

3. For making HTTP GET requests, the function getURL is used with the option ssl.vertifypeer=false, in order to avoid SSL security certificate errors. We have also used the fromJSON() function from the rjson library to convert the string response to an object.
4. The postform function provides the httpheader option to set up the header. We have used it to set up the "Content-Type" header parameter but it would also be possible to use it for setting up basic authentication. The other functions are similar to what we have being doing on the other examples.
5. The content of the main.r file is:

```
MainScript<- function() {

source('c:\\piwebapi.r');

library(RCurl);

library(rjson);

citiesName = GetCitiesName();

attributesName = GetAttributesName();

SendValue(citiesName[1],attributesName[2],50);

}
```

On R, you should load both libraries before running the script through the library() method.

If you are having issues with RCurl, this FAQ page provides some useful information. More information about other options for RCurl function are available on this page.

4. Programming with Bulk Calls

One of the new enhancements of PI Web API 2015 R2 is the ability to make bulk calls (read and write) for multiple streams (PI Points and Attribute) through a single HTTP request. Nevertheless, a prerequisite for using this feature is to know the WebID of each stream in advance. This means if you want to make a single bulk call to get recorded values from three streams, you need to make 3 HTTP requests previously in order to get the streams WebIDs.

In this section, this issue will be overcome using JavaScript and C#.

JavaScript

A solution for making bulk calls is to use the $.when() method from jQuery as shown below:

```
        var ajaxPt1 = getPIPointWebId(piDataArchiveName, piPointNames[0], null,
errorCallBack);
        var ajaxPt2 = getPIPointWebId(piDataArchiveName, piPointNames[1], null,
errorCallBack);
        var ajaxPt3 = getPIPointWebId(piDataArchiveName, piPointNames[2], null,
errorCallBack);
        var data = [];
        $.when(ajaxPt1, ajaxPt2, ajaxPt3).done(function (r1, r2, r3) {
            var results = new Array(3);
            results[0] = r1[0];
            results[1] = r2[0];
            results[2] = r3[0];
            ……
        });
```

This example was taken from a PI Developers Club blog post. Please click here to read the complete article about this topic.

The variables ajaxPt1, ajaxPt3, ajaxPt3 are ajax functions that will get the WebIDs from the PI Points of the array. The method when() will execute them and it will continue with the done() method as soon as all the requests have their execution finished. The results from those ajax requests are available on the r1, r2 and r3 variables which contain the WebIDs to proceed with the bulk calls.

C#

The natural question for a .NET developer is: how to do that in C#?

First, we define a List<string> with all the PI Point names and a dynamic object which will store the final response containing all the compressed values, Then, we create an array of tasks responsible for getting the WebIDs from the streams. Each task will get the WebID from each stream.

This example was taken from a PI Developers Club blog post. Please click here to read the complete article about this topic.

When all tasks have finished storing the WebIDs from the streams, the url for getting the compressed values needs to be built. We create another task called continuation which will be executed when all of the antecedent tasks have their execution finished. The continuation task will access the Result property from each antecedent Task for getting the WebIDs. Adding the start time and end time, all the inputs are available to complete this task by calling the MakeRequest(url) method again. Finally, the antecedent tasks are started and the main thread waits for the continuation task to complete. Please refer to the code snippet below:

```
// create an array of pi point names
            List<string> piPointNames = new List<string>() { "sinusoid", "sinusoidu",
"cdt160" };

            //this variable will store the final response with the compressed values
from all the PI Points from the array
            dynamic finalResponse = null;

            // create an array of tasks
            Task<string>[] tasks = new Task<string>[piPointNames.Count];

            for (int i = 0; i < piPointNames.Count; i++)
            {

                // create a new task
                tasks[i] = new Task<string>((piPointName) =>
                    {

                        //get the webId from the selected PI Point
                        string url = @"https://marc-web-
sql.marc.net/piwebapi/points?path=\\marc-pi2014\" + piPointName;
                        dynamic response = MakeRequest(url);
                        string webId = response.WebId.Value.ToString();

                        //the webIds will be available by accessing the property
Task.Result
                        return webId;
                    }, piPointNames[i]);
            }

            // set up a multitask continuation
            Task continuation = Task.Factory.ContinueWhenAll(tasks, antecedents =>
            {
                //in order to generate an url with all webids to get compressed values
from all PI Points within the array, the antecents tasks need be accessed
                string webIdsSection = string.Empty;
                foreach (Task<string> t in antecedents)
                {
                    webIdsSection += "&webid=" + t.Result;
                }
                //then, we can generate the url with the start time and end time
                string url = @"https://marc-web-
sql.marc.net/piwebapi/streamsets/recorded?" + webIdsSection.Substring(1) +
"&startTime=*-200d&endTime=*";
                finalResponse = MakeRequest(url);
```

```
            });

            // start the antecedent tasks
            foreach (Task t in tasks)
            {
                t.Start();
            }

            continuation.Wait();

            Console.WriteLine(finalResponse);
            Console.WriteLine("Press enter to finish");
            Console.ReadLine();
        }

        internal static dynamic MakeRequest(string url)
        {
            WebRequest request = WebRequest.Create(url);

            //Setting up this property will add the Basic Authentication on the header
of the HTTP request
            request.Credentials = new NetworkCredential("marc.adm", "xxxxx");
            request.ContentType = "application/json";
            WebResponse response = request.GetResponse();
            using (StreamReader sw = new StreamReader(response.GetResponseStream()))
            {
                using (JsonTextReader reader = new JsonTextReader(sw))
                {
                    return JObject.ReadFrom(reader);
                }
            }
        }
```

5. Practical Use Cases

On this chapter we will focus on three practical uses cases of applications developed with PI Web API:

- Migrating C# from PI AF SDK to PI Web API
- Single-Pages Applications with AngularJS (future version)
- Android Apps using Android SDK (future version)

All the projects available are accessible by download the supporting files.

Migrating C# applications from PI AF SDK to PI Web API

With the PI Web API release, there might be an interest in migrating a C# application using PI AF SDK to PI Web API. The main reason would be that there is no need to install any OSIsoft product on the client machine if the application connects to the PI System through PI Web API. This occur since all communication is done through RESTful calls. Nevertheless, PI Web API wraps PI AF SDK which means that using PI AF SDK directly on your application should result in better performance.

Below you can find a simple console application using PI AF SDK. This application creates the AF database used in the previous chapters.

PI AF SDK application

```
using OSIsoft.AF;
using OSIsoft.AF.Asset;
using OSIsoft.AF.UnitsOfMeasure;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace DatabaseCreationWithPIAFSDK
{
    class Program
    {
        private static PISystem myPISystem = null;
        private static string PIServerName = string.Empty;
        static void Main(string[] args)
        {
            myPISystem = new PISystems()["MARC-PI2014"];
            PIServerName = "MARC-PI2014";
            AFDatabase myDb = myPISystem.Databases["PIWebAPIWhitePaperSampleDb"];
            if (myDb == null)
            {
                myDb = myPISystem.Databases.Add("PIWebAPIWhitePaperSampleDb");
            }
```

```
            AFElementTemplate myCityTemplate = myDb.ElementTemplates["CityTemplate"];
            if (myCityTemplate == null)
            {
                myCityTemplate = myDb.ElementTemplates.Add("Cities");
            }


            UOM degreeC =
myPISystem.UOMDatabase.UOMClasses["Temperature"].UOMs["degree Celsius"];
            UOM speedKmph = myPISystem.UOMDatabase.UOMClasses["Speed"].UOMs["kilometer
per hour"];
            UOM millibar =
myPISystem.UOMDatabase.UOMClasses["Pressure"].UOMs["milibar"];
            UOM percent = myPISystem.UOMDatabase.UOMClasses["Ratio"].UOMs["percent"];
            UOM kilometres =
myPISystem.UOMDatabase.UOMClasses["Length"].UOMs["kilometer"];


            CreateAttributeTemplate("Cloud Cover", myCityTemplate, typeof(int),
percent);
            CreateAttributeTemplate("Humidity", myCityTemplate, typeof(double),
percent);
            CreateAttributeTemplate("Pressure", myCityTemplate, typeof(int),
millibar);
            CreateAttributeTemplate("Temperature", myCityTemplate, typeof(int),
degreeC);
            CreateAttributeTemplate("Visibility", myCityTemplate, typeof(int),
kilometres);
            CreateAttributeTemplate("Wind Speed", myCityTemplate, typeof(int),
speedKmph);

            AFElement cities = myDb.Elements.Add("Cities");
            cities.Elements.Add("Chicago", myCityTemplate);
            cities.Elements.Add("Los Angeles", myCityTemplate);
            cities.Elements.Add("New York", myCityTemplate);
            cities.Elements.Add("San Francisco", myCityTemplate);
            cities.Elements.Add("Washington", myCityTemplate);
            myDb.CheckIn();
        }

        public static void CreateAttributeTemplate(string name, AFElementTemplate
elementTemplate, Type attributeType, UOM attributeUOM)
        {
            AFAttributeTemplate myAttributeTemplate =
elementTemplate.AttributeTemplates[name];
            if (myAttributeTemplate == null)
            {
                myAttributeTemplate = elementTemplate.AttributeTemplates.Add(name);
            }
            myAttributeTemplate.DataReferencePlugIn =
myPISystem.DataReferencePlugIns["PI Point"];
            myAttributeTemplate.Type = attributeType;
            myAttributeTemplate.DefaultUOM = attributeUOM;
            if (myAttributeTemplate.DefaultUOM == null)
            {
                myAttributeTemplate.ConfigString = @"\\" + PIServerName +
@"\%Element%_%Attribute%;";
            }
            else
            {
                myAttributeTemplate.ConfigString = @"\\" + PIServerName +
@"\%Element%_%Attribute%;UOM=" + myAttributeTemplate.DefaultUOM.Abbreviation;
            }
```

```
        }
    }
}
```

The workflow of this program is quite simple:

1. Connect to the PI System
2. Create a new database if it is not found.
3. Create the city element template.
4. Add six attributes templates to the city element template.
5. Create the cities element on the root.
6. Create six elements derived from the city element template under the cities element.

PI Web API application

Let's take a look now at the PI Web API version. The content of the Program.cs file is below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseCreationForPIWebAPI
{
    class Program
    {
        static void Main(string[] args)
        {
            string piSystemName = "MARC-PI2014";
            string piServerName = "MARC-PI2014";
            string afDatabaseName = "PIWebAPIWhitePaperSampleDb2";
            string afElementTemplateName = "CityTemplate";
            string assetDatabasesWebId =
PIWebAPIWrapper.GetAssetDatabaseWebId(piSystemName, afDatabaseName);
            int statusCode = -1;
            if (assetDatabasesWebId==null)
            {
               statusCode = PIWebAPIWrapper.CreateAssetDatabase(piSystemName,
afDatabaseName);
            }
            assetDatabasesWebId = PIWebAPIWrapper.GetAssetDatabaseWebId(piSystemName,
afDatabaseName);

            statusCode = PIWebAPIWrapper.CreateElementTemplate(afElementTemplateName,
assetDatabasesWebId);
            statusCode = PIWebAPIWrapper.CreateElement("Cities", assetDatabasesWebId,
true);
```

```
            string elementTemplateWebId =
PIWebAPIWrapper.GetElementTemplateWebId(piSystemName, afDatabaseName,
afElementTemplateName);
            string citiesElementWebId = PIWebAPIWrapper.GetElementWebId(@"\\" +
piSystemName + @"\" + afDatabaseName + @"\Cities");


            statusCode = PIWebAPIWrapper.CreateAttributeTemplate(elementTemplateWebId,
"Cloud Cover", "percent", piServerName, "Int32");
            statusCode = PIWebAPIWrapper.CreateAttributeTemplate(elementTemplateWebId,
"Humidity", "percent", piServerName, "Double");
            statusCode = PIWebAPIWrapper.CreateAttributeTemplate(elementTemplateWebId,
"Pressure", "milibar", piServerName, "Int32");
            statusCode = PIWebAPIWrapper.CreateAttributeTemplate(elementTemplateWebId,
"Temperature", "degree Celsius", piServerName, "Int32");
            statusCode = PIWebAPIWrapper.CreateAttributeTemplate(elementTemplateWebId,
"Visibility", "kilometer", piServerName, "Int32");
            statusCode = PIWebAPIWrapper.CreateAttributeTemplate(elementTemplateWebId,
"Wind Speed", "kilometer per hour", piServerName, "Int32");

            statusCode = PIWebAPIWrapper.CreateElement("Chicago", citiesElementWebId,
false, afElementTemplateName);
            statusCode = PIWebAPIWrapper.CreateElement("Los Angeles",
citiesElementWebId, false, afElementTemplateName);
            statusCode = PIWebAPIWrapper.CreateElement("New York", citiesElementWebId,
false, afElementTemplateName);
            statusCode = PIWebAPIWrapper.CreateElement("San Francisco",
citiesElementWebId, false, afElementTemplateName);
            statusCode = PIWebAPIWrapper.CreateElement("Washington",
citiesElementWebId, false, afElementTemplateName);
        }
    }
}
```

We have created a PIWebAPIWrapper class to replace some methods of the PI AF SDK. The methods created on this class are:

- GetAssetDatabaseWebId
- GetPISystemWebId
- GetElementTemplateWebId
- GetElementWebId
- CreateAssetDatabase
- CreateElement
- CreateElementTemplate
- CreateAttributeTemplate

The four methods that start with Get return the WebID of some objects to be used on other methods. The other four methods that create objects return the status code. For this testing purposes, we are not checking if the status code returned is 201, but in your program this is something recommended to implement in order to make sure that the request was executed successfully.

The content of the PIWebAPIWrapper.cs is:

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading.Tasks;

namespace DatabaseCreationForPIWebAPI
{
    public class PIWebAPIWrapper
    {
        private static string base_url = "https://marc-web-sql.marc.net/piwebapi/";

        public static string GetAssetDatabaseWebId(string piSystemName, string
afDatabaseName)
        {
            dynamic jsonObj = jsonObj = MakeGetRequest(base_url +
@"assetdatabases?path=\\" + piSystemName + @"\" + afDatabaseName);
            string json = jsonObj.ToString();
            int intResult = 0;
            bool result = Int32.TryParse(json, out intResult);
            if (result == false)
            {
                return jsonObj.WebId.Value;
            }
            else
            {
                return null;
            }
        }

        public static string GetPISystemWebId(string piSystemName)
        {
            dynamic jsonObj = jsonObj = MakeGetRequest(base_url +
@"assetservers?path=\\" + piSystemName);
            return jsonObj.WebId.Value;
        }

        public static int CreateAssetDatabase(string piSystemName, string
afDatabaseName)
        {
            string piSystemWebId = GetPISystemWebId(piSystemName);
            int statusCode = -1;
            string url = base_url + "assetservers/" + piSystemWebId +
"/assetdatabases";
            string postData = "{\"Name\": \"" + afDatabaseName + "\", \"Description\":
\"Database of the PI Web API white paper\"}";
            MakePostRequest(url, postData, out statusCode);
            return statusCode;
        }

        public static string GetElementTemplateWebId(string piSystemName, string
afDatabaseName, string afElementTemplateName)
        {
            string elementTemplatePath = @"\\" + piSystemName + @"\" + afDatabaseName
+ @"\ElementTemplates[" + afElementTemplateName + "]";
            dynamic jsonObj = MakeGetRequest(base_url + @"elementtemplates?path=" +
elementTemplatePath);
```

```
            return jsonObj.WebId.Value;
        }

        public static string GetElementWebId(string elementPath)
        {
            dynamic jsonObj = MakeGetRequest(base_url + @"elements?path=" +
elementPath);
            return jsonObj.WebId.Value;
        }

        public static int CreateElement(string afElementName, string webId, bool
onRoot, string templateName = null)
        {
            int statusCode = -1;
            string url = string.Empty;
            string postData = "{\"Name\": \"" + afElementName + "\"}";
            if (templateName != null)
            {
                postData = "{\"Name\": \"" + afElementName + "\", \"TemplateName\":
\"" + templateName + "\"}";
            }
            if (onRoot == true)
            {
                url = base_url + "assetdatabases/" + webId + "/elements";
            }
            else
            {
                url = base_url + "elements/" + webId + "/elements";
            }
            MakePostRequest(url, postData, out statusCode);
            return statusCode;
        }




        public static int CreateElementTemplate(string afElementTemplateName, string
assetdatabasesWebId)
        {
            int statusCode = -1;
            string postData = "{\"Name\": \"" + afElementTemplateName + "\"}";
            string url = base_url + "assetdatabases/" + assetdatabasesWebId +
"/elementtemplates";
            MakePostRequest(url, postData, out statusCode);
            return statusCode;
        }


        public static int CreateAttributeTemplate(string elementTemplateWebId, string
afAttributeTemplateName, string uomString, string piServerName, string attributeType)
        {
            int statusCode = -1;
            string postData = "{\"Name\": \"" + afAttributeTemplateName +
"\",\"Type\": \"" + attributeType + "\",\"DefaultUnitsName\": \"" + uomString +
"\",\"DataReferencePlugIn\": \"PI Point\",  \"ConfigString\": \"\\\\\\\\\" +
piServerName + "\\\\%Element%_%Attribute%\"}";
            string url = base_url + "elementtemplates/" + elementTemplateWebId +
"/attributetemplates";
            MakePostRequest(url, postData, out statusCode);
            return statusCode;
        }


        internal static dynamic MakeGetRequest(string url)
```

```
        {
            WebRequest request = WebRequest.Create(url);
            //Basic Authentication
            request.Credentials = new NetworkCredential("username", "password");

            //Kerberos
            //request.Credentials = CredentialCache.DefaultCredentials;
            WebResponse response = null;
            try
            {
                response = request.GetResponse();
            }
            catch (WebException ex)
            {
                int statusCode = (int)((HttpWebResponse)ex.Response).StatusCode;
                return (dynamic)statusCode;
            }

            using (StreamReader sw = new StreamReader(response.GetResponseStream()))
            {
                using (JsonTextReader reader = new JsonTextReader(sw))
                {
                    return JObject.ReadFrom(reader);
                }
            }
        }


        internal static void MakePostRequest(string url, string postData, out int
statusCode)
        {
            WebRequest request = WebRequest.Create(url);
            ((HttpWebRequest)request).UserAgent = ".NET Framework Example Client";
            request.Method = "POST";

            //Basic Authentication
            request.Credentials = new NetworkCredential("username", "password");

            //Kerberos
            //request.Credentials = CredentialCache.DefaultCredentials;

            request.ContentType = "application/json";
            byte[] byteArray = Encoding.UTF8.GetBytes(postData);
            request.ContentLength = byteArray.Length;
            Stream dataStream = request.GetRequestStream();
            dataStream.Write(byteArray, 0, byteArray.Length);
            dataStream.Close();
            WebResponse response = request.GetResponse();
            statusCode =
Convert.ToInt32(((System.Net.HttpWebResponse)(response)).StatusCode);
        }
    }
}
```

You probably have realized that we are using similar MakeGetRequest and MakePostRequest methods from the previous chapter.

PI Web API 2015 R2 does not provide all functionalities of PI AF SDK 2015. Nevertheless, it has the most important ones and the development team keeps adding new functionalities to the

upcoming releases. In case you don’t want to install PI AF Client on client machines, migrating your code to PI Web API is a feasible option.

6. Final Comments

On the upcoming versions of this white paper, two new topics shall be added:

- Developing a simple web application using AngularJs to manage elements and attributes.
- Developing smartphone native apps using Android SDK.

We hope you have enjoyed and learned a lot of PI Web API development for different programming languages.

Stay tuned for the next release!

Revision History

Date	Revision No.
22-Dec-14	Initial draft by Marcos Loeff
30-Sep-15	Second revision by Marcos Loeff