

PI World 2019 Lab

Modernizing PI SDK Tag-based Applications



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

© 2019 by OSIssoft, LLC. All rights reserved.

OSIssoft, the OSIssoft logo and logotype, Analytics, PI ProcessBook, PI DataLink, ProcessPoint, Asset Framework (AF), IT Monitor, MCN Health Monitor, PI System, PI ActiveView, PI ACE, PI AlarmView, PI BatchView, PI Vision, PI Data Services, Event Frames, PI Manual Logger, PI ProfileView, PI WebParts, ProTRAQ, RLINK, RtAnalytics, RtBaseline, RtPortal, RtPM, RtReports and RtWebParts are all trademarks of OSIssoft, LLC. All other trademarks or trade names used herein are the property of their respective owners.

U.S. GOVERNMENT RIGHTS

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the OSIssoft, LLC license agreement and as provided in DFARS 227.7202, DFARS 252.227-7013, FAR 12.212, FAR 52.227, as applicable. OSIssoft, LLC.

Published: March 20, 2019

Table of Contents

Contents

1. Introduction	4
1.1 PI SDK and COM Are Old Technologies.....	4
1.2 Adding New Life to Tag-based Applications	4
1.3 Excuses Against AF SDK.....	5
2. Learning Environment.....	6
2.1 Installed Software	6
2.2 Tag Naming Patterns.....	6
2.3 Where to Find the Exercises	6
2.4 Live Library Help.....	8
3. Common Tasks for Each Exercise	11
3.1 Set Startup Project.....	11
3.2 Set Target .NET Framework	12
3.3 Swap PI SDK References with AF SDK	12
3.4 Add C# <i>using</i> or VB <i>Imports</i> to Your Code	13
3.5 Remove STAThread Attribute	14
3.6 Remove COM Housekeeping	14
3.7 PI SDK Performance Versus AF SDK	15
4. The Exercises.....	17
4.1 Exercise 1 – Connect to a PI Data Archive and Read Data.....	19
4.2 Exercise 2 – Reading Data from a List of PI Points.....	23
4.3 Exercise 3 – Writing Data in Bulk	27
4.4 Exercise 4 – Execute Searches for PI Points and Retrieve Summary Data.....	31
Appendix 1 – Bonus Applications.....	35
Tag Creation Bonus with AF SDK.....	35
PI Data Archive Metadata Bonus	35
Appendix 2 – C# Cheat Sheet for VB.NET	39
Save the Date!	41

1. Introduction

1.1 PI SDK and COM Are Old Technologies

PI SDK heavily relies upon Microsoft's Component Object Model (COM) and both are considered aging technologies. In 2015, OSIsoft stopped recommending PI SDK for any new development. PI SDK has been in maintenance mode for many years, meaning there are no – and will be no – new features released. All you may expect from PI SDK releases are bug fixes and security patches.

While Component Object Model is efficient at certain low level tasks or things like interprocess communications, many application developers find it difficult to work with. COM applications are cumbersome to write, difficult to debug, sluggish in performance, and prone to memory leaks.

1.2 Adding New Life to Tag-based Applications

A common goal for every developer of tag-based applications would be to have their critical applications continue to perform their intended functions, and not to be an unfortunate casualty of a Microsoft decision to cease support, as they have done for Windows XP, Internet Explorer, Silverlight, etc.

There is a way to extend the life of your tag-based applications. That's the good news. The better news is that your applications will be more secure and perform faster too. These benefits may all be realized by porting your applications away from COM and to managed .NET Framework. This means migrating the application from PI SDK to AF SDK.

Let's consider some of the benefits.

- AF SDK does not use a COM data access layer when communicating with the PI Data Archive. This avoids STA/MTA issues that exist when developing .NET applications with the PI SDK.
- COM objects also no longer need to be marshalled to equivalent .NET data types, which improves performance.
- Finally, the experience of developing applications that target PI using the AF SDK is more pleasant, since it was designed expressly for .NET developers. Classes such as `PINamedValues` that were not so easy to work with via COM Interop have been replaced by the more ubiquitous .NET dictionary objects.

1.3 Excuses Against AF SDK

Here are the top 3 excuses against using AF SDK:

1. "I don't have an AF database."
2. "IT won't let me install it."
3. "IT won't let me use Visual Studio."

Let's dispel the biggest myth about AF SDK: that you must have an AF Server and or AF database. This is not true. Obviously, if you were wanting to develop an *Asset*-based application, then you would need an asset hierarchy. But to develop a *Tag*-based application, such as what you have done in the past with PI SDK, you absolutely do not need an AF Server or database. Your application may interact directly with a PI Data Archive and PI points without ever touching an asset hierarchy. The exercises in this lab will clearly prove that.

A claim that IT will not let you install AF SDK is hard to fathom. Since PI Server 2012, the AF Client has been a standard part of the PI Data Archive setup. If your company has upgraded the PI Server since 2012, you should already have AF SDK installed. However, if your company has not upgraded since 2012, then perhaps modernizing the infrastructure should be a higher priority than modernizing a Tag-based application that runs within that infrastructure.

A credible excuse is that IT restricts access to Visual Studio. Different IT departments possess different reasons for doing so. To address a few of these:

- If IT wants to control cost of software deployments, consider using Visual Studio Code, which is free and its license allows both personal and commercial development.
- If IT is worried about security, point out that managed .NET has better security than unmanaged COM, and therefore an AF SDK Tag-based application is much more secure than a PI SDK one.
- If IT only wants qualified developers to use Visual Studio, the argument rests fully on your shoulders to convince them of your qualifications as a capable application developer.

If IT still refuses to give you Visual Studio after you and your manager have presented your strongest justification, you may still use AF SDK with PowerShell. Note, however, PowerShell is not covered within this lab.

2. Learning Environment

2.1 Installed Software

Your VM has been setup with the following:

- PI Data Archive 2018 SP2, which is the latest production release
- PI SMT
- PI SDK client, 32 and 64 bit (the exercises use 64 bit)
- PI SDK Interop libraries for .NET
- AF SDK client (.NET 4.0 compatible)
- Visual Studio 2017 Community Edition

As this lab will work only with tag-based applications, take note that your VM lacks the following since they are not needed:

- An AF Database
- An AF Server
- SQL Server

The PI Buffer service is loaded. However, since the Visual Studio applications reside on the same VM as the PI Data Archive, buffering is not enabled. See relevant comments about this in the solution for Exercise 3, which performs bulk writes to the PI Data Archive.

2.2 Tag Naming Patterns

The tags used in the exercises have the following naming patterns and counts:

Name Pattern	Task	Tags
Exercise 2.*	Bulk reads	260
Exercise 3.*	Bulk writes	1000
Exercise 4.*	Bulk summary	314

2.3 Where to Find the Exercises

All the Exercises for this lab can be found in a single Visual Studio Solution file. This lab offers versions in C# as well as VB.NET:

*C:\Modernizing Tag-based Applications**CSharp**\Modernizing Tag-based Applications.sln*

*C:\Modernizing Tag-based Applications**VB**\Modernizing Tag-based Applications.sln*

There is also a convenient shortcut on your VM's desktop, pointing to the folder:

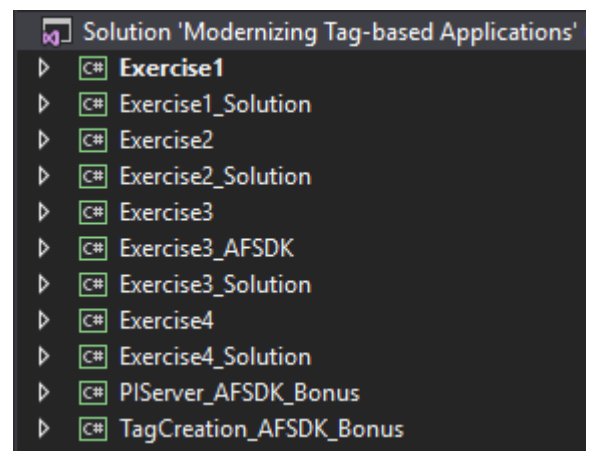
C:\Modernizing Tag-based Applications

Each Exercise has a separate project for the challenge, as well one possible solution. As you work your way through the lab, you can set the project that is being debugged via right-clicking in the Visual Studio Solutions tab and selecting **Set as Startup Project**.

Each exercise is invoked from the **Main** in a similar manner, and your coding efforts can begin in the routine **Exercise1**, **Exercise2**, etc.

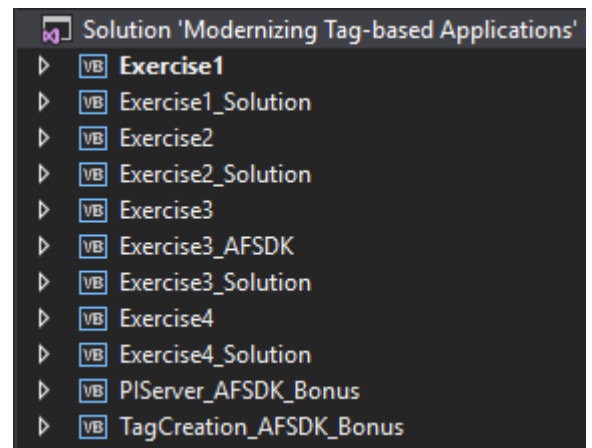
C#:

```
static void Main(string[] args)
{
    try
    {
        Exercis1();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    Console.WriteLine();
    Console.WriteLine("Press Enter to Exit");
    Console.ReadLine();
}
```



VB.NET

```
Sub Main()
    Try
        Exercis1()
    Catch ex As Exception
        Console.WriteLine(ex.Message)
    End Try
    Console.WriteLine()
    Console.WriteLine("Press Enter to Exit")
    Console.ReadLine()
End Sub
```



Each project contains hints via URL links to Live Library Help. (Although jumping right to the respective solution project is the ultimate hint ☺.)

2.4 Live Library Help

Installing the AF Client will also setup the **AFSDK.CHM** help file. Live Library is the online equivalent of the local CHM file. While the PI SDK and the AF SDK are similar, there are differences with which you will become familiar through this exercises and browsing the help.

1. Familiarize yourself with the AF SDK Help. You may use the online help offered by Live Library or help via the local CHM file. The lessons here utilize Live Library.

For Live Library, open a web browser and enter the URL:

<https://livelibrary.osisoft.com/LiveLibrary/>

Scroll to the section listed as Developer Technologies. Click on AF SDK Reference.

Developer Technologies

AF SDK Reference

Programmer reference for AF SDK

[View Documentation](#) 

AF SDK Getting Started

Explore AF SDK entry-level concepts and examples

[View Documentation](#)

PI Web API Reference

Programmer reference for PI Web API

[View Documentation](#)

PI OLEDB Enterprise

Use SQL to access PI System data with an OLEDB provider

[View Documentation](#)

2. For example, the **Overview / PI SDK Equivalent** and **Examples / Connecting to PI Data Archive** will be useful starting with Exercise 1.

▲ AF SDK Reference	←	
▲ Overview	←	
▷ What's New		
• .NET 4.5 Differences		
• PI SDK Equivalents	←	
• AF SDK Buffering		
• Future Data		
• Path Syntax Overview		
• PIPoint Query Syntax Overview		
• Search Overview		
• Security Formats		
• Threading Overview		
• List / Bulk Data Methods Overview		
• Data Call Error Handling		
▷ Analysis Execution		
▷ Hierarchies		
▷ Examples	←	
▷ Namespaces		
• Copyright Notice		
		▷ AF SDK Reference
		▲ Examples
		• Asynchronous Data Methods
		▷ Bulk Load Example
		• CheckIn Example
		• Connecting to a PI AF Server
		• Connecting to a PI Data Archive
		• Event Example
		• Search Example
		• Search Aggregation Example
		• Version Example

3. Common Tasks for Each Exercise

Each exercise has a different primary task, but they share many common sub-tasks. These sub-tasks may need to be performed for each of the 4 exercises. This assumes you have loaded Visual Studio 2017, and opened the solution file **Modernizing Tag-based Applications.sln**.

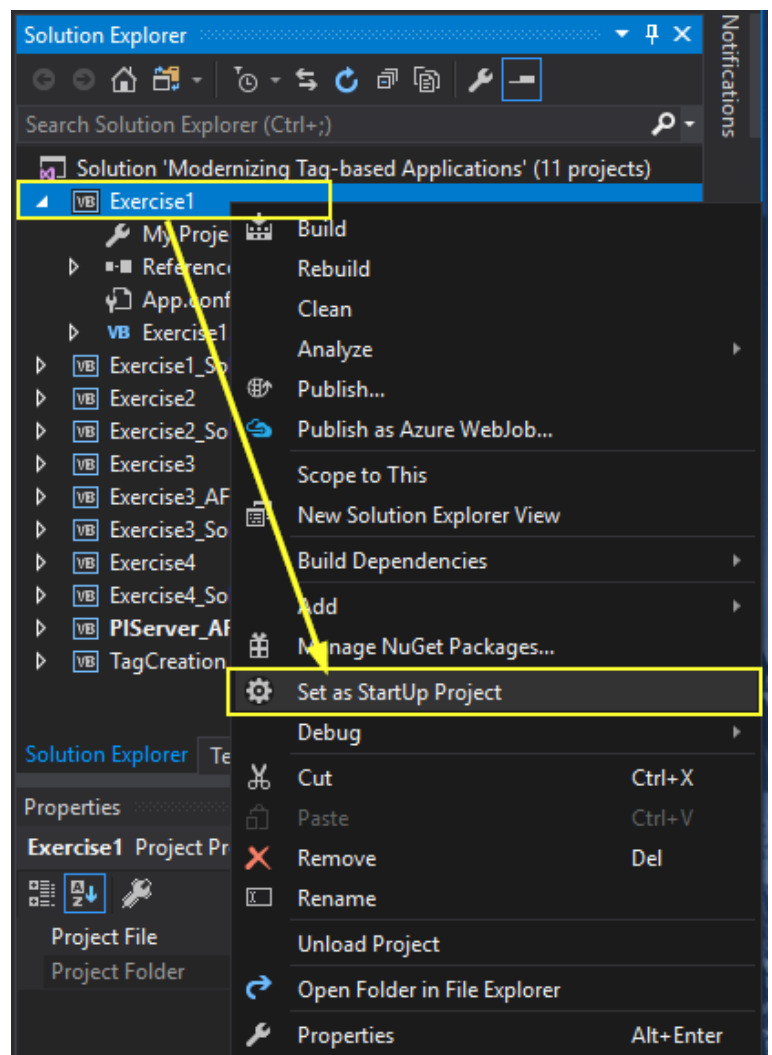
3.1 Set Startup Project

Before running a specific exercise or its solution, be sure to right-click on the project in the Solution Explorer pane and select **Set as Startup Project**.

See illustration to the right for an example of selecting Exercise1 as the startup project.

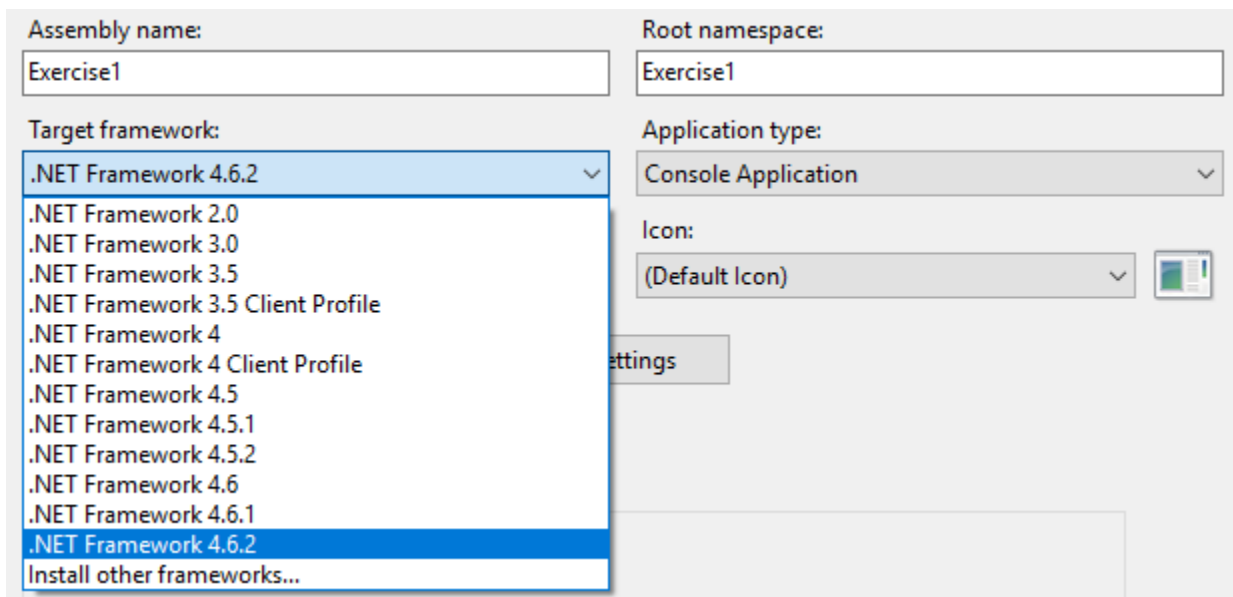
Later, whenever you click the green **Start** button at the top of Visual Studio's menu bar, the designated startup project will run for the current exercise.

The solution has 11 different projects, so be sure you are on the correct one for the exercise at hand.



3.2 Set Target .NET Framework

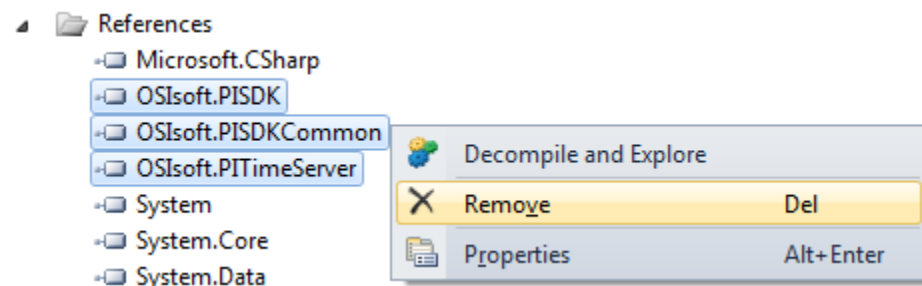
The specific release of AF SDK determines the minimum .NET Framework that your application must target. This is usually found in the PI AF Release Notes. For this lab, the target framework should be .NET Framework 4.6.2. Note: this has already been set for you.



The OSIssoft.AF.PI namespace is only available for .NET 4.x Framework or higher.

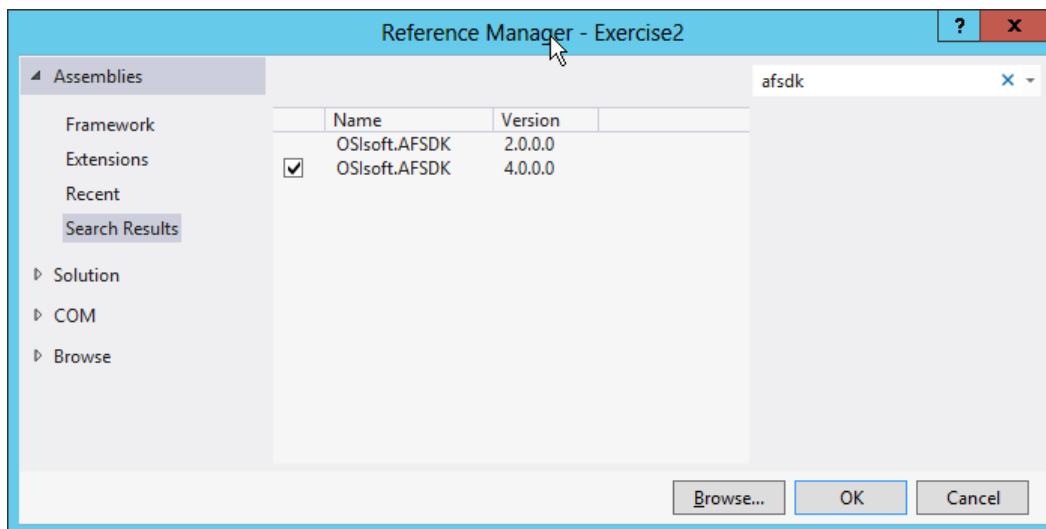
3.3 Swap PI SDK References with AF SDK

Remove the references to the PI SDK and add a reference to the AF SDK 4.0.0.0. To remove PI SDK from your project:



Then you may add AF SDK references. Note: although the AF SDK library has already been referenced for you, here are the steps for future reference:

Please note that it takes some time for Visual Studio to fully populate the list of components in the .NET tab of the Add Reference dialog, and often the 2.0 version of the AF SDK will show up first. **Wait to select the 4.0 version.**



Be sure to click on the check box to select the 4.0.0.0 version *before* clicking the OK button. A common mistake is merely to highlight the row for 4.0.0.0 and then click OK; doing so will not add any references.

3.4 Add C# *using* or VB *Imports* to Your Code

Begin converting the source code from a PI SDK solution to an AF SDK solution. It will be easiest if you add C# *using* or VB *Imports* statements for AF to the top of each program:

C#	VB
<code>using OSIsoft.AF.Data;</code>	<code>Imports OSIsoft.AF.Data</code>
<code>using OSIsoft.AF.PI;</code>	<code>Imports OSIsoft.AF.PI</code>
<code>using OSIsoft.AF.Asset;</code>	<code>Imports OSIsoft.AF.Asset</code>
<code>using OSIsoft.AF.Time;</code>	<code>Imports OSIsoft.AF.Time</code>

3.5 Remove STAThread Attribute

COM has an affinity for single thread apartments (STA). To put it another way, a PI SDK application running in a multi-threaded apartment (MTA) would be slower than its STA counterpart. However, .NET is happier and performs better with MTA. Thus, you should delete the **STAThread** attribute for AF SDK.

```
[STAThread] // C# - Delete attribute on left
static void Main(string[] args)
```

```
<STAThread()> ' VB - Delete attribute on left
Sub Main(ByVal args() As String)
```

3.6 Remove COM Housekeeping

AF SDK uses managed .NET code, so there is no need for the extra lines of “housekeeping” code to release COM objects. You may shorten the code by removing any snippets associated with the pseudo-code below:

```
Marshal.ReleaseComObject(variable)

Marshal.FinalReleaseComObject(variable)

GC.KeepAlive(variable)
```

To elaborate on “snippets associated”, this does not mean just delete the one line. For instance, here a block needs to be deleted since the entire block is associated with COM housekeeping:

```
// C# - COM Housekeeping Block
if (values != null)
{
    Marshal.ReleaseComObject(values);
    values = null;
}
```

```
' VB - COM Housekeeping Block
If values IsNot Nothing Then
    Marshal.ReleaseComObject(values)
    values = Nothing
End If
```

3.7 PI SDK Performance Versus AF SDK

Exercises 2 through 4 use a **Diagnostics.Stopwatch** to measure performance. You may try a performance test before you modify any of the code for these exercises. To do that:

1. Set Exercise**N** as the Startup Project, where “**N**” is the exercise number.
 - a. This uses PI SDK.
 - b. Keep in mind this may have been a cold run without previous caching.
2. Set Exercise**N**_Solution as the Startup Project.
 - a. This uses AF SDK.
 - b. Run the exercise and note the elapsed time displayed in the console window.
 - c. Keep in mind this would have been a warm run with caching from the previous run.
3. Once again set Exercise**N** as the Startup Project.
 - a. This once again uses PI SDK.
 - b. Run the exercise and note the elapsed time displayed in the console window.
 - c. Keep in mind this would too be a warm run.
 - d. Finally, compare the timings here with the AF SDK run in step 2 above.

What you will observe is the AF SDK examples run faster than the PI SDK equivalent. If you do not see this, be sure the AF SDK application does not use the **STAThread** attribute.

4. The Exercises

The challenge presented in the four lab exercises will be to migrate several tag-based applications from PI SDK to AF SDK. Neither an AF Server nor an AF Database will be used as all requests will go directly to a PI Data Archive.

1. Connect to a PI Data Archive and obtain data using the AF SDK
2. Retrieve data for a list of PI Points using bulk methods for better performance
3. Write data using bulk methods for better performance
4. Execute searches for PI Points and retrieving summary data

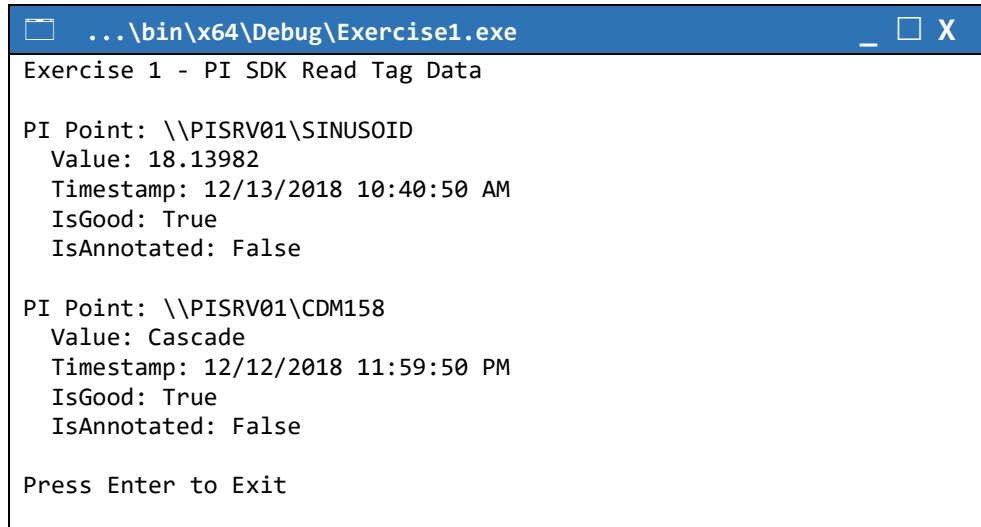
As a bonus, there are 2 small applets that provide further learning examples of using the AF SDK for tag-based applications. Since these applets are not exercises, an equivalent PI SDK example is not presented.

1. Programmatically create all “Exercise **N**.” tags used in these exercises.
2. Querying a PI Data Archive for metadata, such as information about its Point Classes or PI State Sets.

4.1 Exercise 1 – Connect to a PI Data Archive and Read Data

The existing project is a Console application which uses the PI SDK to connect to the default PI Data Archive, and then read and display a PI Point current value along with a PI Point archive value. Before beginning the conversion to the AF SDK, you may wish to run the program in its current state to see the expected output.

1. Compile the application and run it. The output should look like the figure below.



```

... \bin\x64\Debug\Exercise1.exe
Exercise 1 - PI SDK Read Tag Data

PI Point: \\PISRV01\SINUSOID
Value: 18.13982
Timestamp: 12/13/2018 10:40:50 AM
IsGood: True
IsAnnotated: False

PI Point: \\PISRV01\CDM158
Value: Cascade
Timestamp: 12/12/2018 11:59:50 PM
IsGood: True
IsAnnotated: False

Press Enter to Exit
  
```

2. Convert the code that connects to the PI Data Archive.
 - a. There is no SDK object in AF; instead, just create the top level **PIServers** collection.
 - b. The AF SDK uses the class **PIServer** instead of **Server**.
 - c. The AF SDK uses the term **Connect** instead of **Open**. Like the PI SDK, connections are opened implicitly as well.
3. Connect to the PI Data Archive.

```

// C# - Connect to the default PI Server
PIServers kst = new PIServers();
PIServer dataArchive = kst.DefaultPIServer;
dataArchive.Connect();

' VB - Connect to the default PI Server
Dim kst As PIServers = New PIServers()
Dim dataArchive As PIServer = kst.DefaultPIServer
dataArchive.Connect()
  
```

Exercise 1

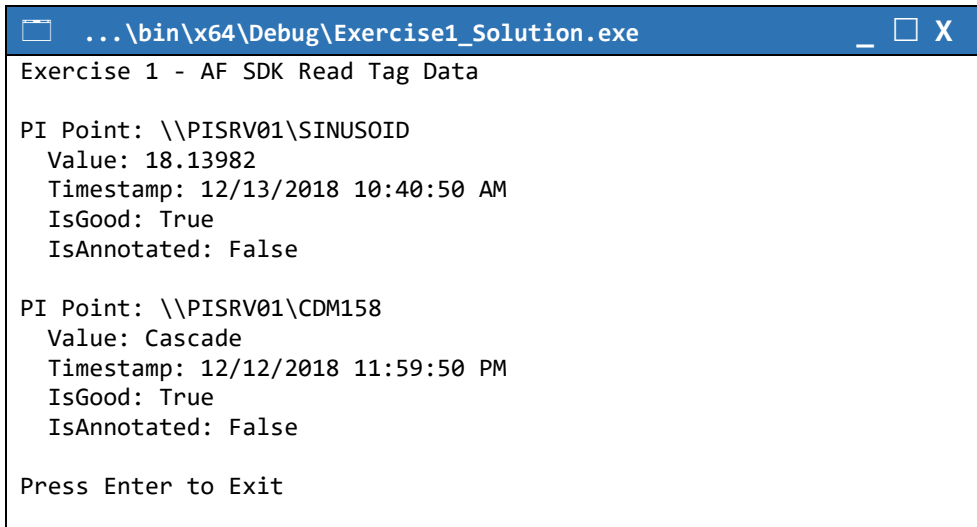
4. Convert the code that retrieves the snapshot - or current value - for the Float32 “sinusoid” PI Point to use the AF SDK’s PI Point object.
 - a. Finds in AF are normally static methods of the object type you are seeking; in this case, **PIPoint.FindPIPoint**. There is no global Points collection as in the PI SDK.
 - b. All data is returned as an **AFValue**.
 - c. Digital States are returned as the type **AFEnumerationValue**, which is a .NET object. Consequently, there’s no need for the If statement used with the PI SDK to determine if the returned value is a Digital State.
 - d. Timestamps are represented by the type **AFTIME**, which has standard .NET DateTime exposure for LocalTime, UtcTime, and UtcSeconds.
 - e. Status bits, such as Annotated, are always available directly from the **AFValue** object.

```
// C# - Retrieve snapshot value of Float32 tag sinusoid
PIPoint point = PIPoint.FindPIPoint(dataArchive, "sinusoid");
AFValue value = point.CurrentValue();
Console.WriteLine("PI Point: {0}", point.GetPath());
Console.WriteLine("  Value: {0}", value.Value);
Console.WriteLine("  Timestamp: {0}", value.Timestamp.LocalTime);
Console.WriteLine("  IsGood: {0}", value.IsGood);
Console.WriteLine("  IsAnnotated: {0}", value.Annotated);
```

5. Convert the code that retrieves an archive value for the “CDM158” PI Point. Though “SINUSOID” is a Float32 tag, and “CDM158” is a digital tag, note the similarity of the code being used.

```
' VB - Retrieve recorded archive of digital tag CDM158
point = PIPoint.FindPIPoint(dataArchive, "cdm158")
Dim today As AFTIME = New AFTIME("t")
value = point.RecordedValue(today, AFRetrievalMode.AtOrBefore)
Console.WriteLine("PI Point: {0}", attr.GetPath())
Console.WriteLine("  Value: {0}", value.Value)
Console.WriteLine("  Timestamp: {0}", value.Timestamp.LocalTime)
Console.WriteLine("  IsGood: {0}", value.IsGood)
Console.WriteLine("  IsAnnotated: {0}", value.Annotated)
```

6. Run your program and validate the output. Source code for the completed solution can be found under **Exercise1_Solution** project. This completes the exercise.



```
...\\bin\\x64\\Debug\\Exercise1_Solution.exe
Exercise 1 - AF SDK Read Tag Data

PI Point: \\PISRV01\\SINUSOID
Value: 18.13982
Timestamp: 12/13/2018 10:40:50 AM
IsGood: True
IsAnnotated: False

PI Point: \\PISRV01\\CDM158
Value: Cascade
Timestamp: 12/12/2018 11:59:50 PM
IsGood: True
IsAnnotated: False

Press Enter to Exit
```

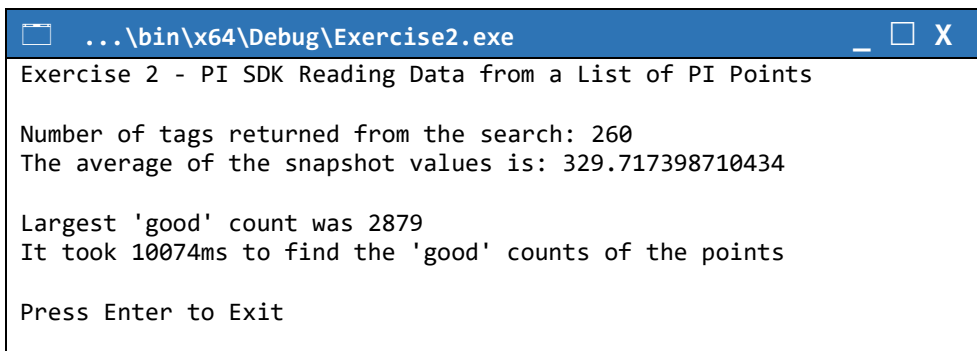

4.2 Exercise 2 – Reading Data from a List of PI Points

In this exercise, you will convert an existing PI SDK program to use the AF SDK as you did in the previous exercise. This application builds a list of PI Points and averages the value of their snapshots. Finally, it retrieves the archive values from the last 24 hours for each tag and finds the largest count of “good” values. The important takeaway should not be the calculations being used but rather on the data retrieval methods being used.

Before you modify any code, you may be interested in a quick performance test. Refer to the section labeled “AF SDK versus PI SDK Performance Tests” to see how to do this.

Steps for this exercise:

1. Make sure Exercise2 is set as the current project. The AF SDK has already been referenced in your project and the appropriate using statements have been added for your convenience.
2. Compile the application and run it. The output should look similar to the figure below.



```

... \bin\x64\Debug\Exercise2.exe
Exercise 2 - PI SDK Reading Data from a List of PI Points

Number of tags returned from the search: 260
The average of the snapshot values is: 329.717398710434

Largest 'good' count was 2879
It took 10074ms to find the 'good' counts of the points

Press Enter to Exit

```

3. Building upon what was learned in Exercise 1, shorten the code that sets to the PI Data Archive. We will use an implicit connection.

```

// C# - Get the default PI Server
PIServer dataArchive = new PIServers().DefaultPIServer;

' VB - Get the default PI Server
Dim dataArchive As PIServer = New PIServers().DefaultPIServer

```

4. Use the static **FindPIPoints** method with the *nameFilter* overload on the **PIPoint** object to search for the PI Points whose tag starts with “Exercise 2.*” Pass the resulting **IEnumerable** into the constructor of **PIPointList** to build your **PIPointList**.

Exercise 2

```
// C# - Get all points for starting with "Exercise 2.*"
IEnumerable<PIPoint> searchResults = PIPoint.FindPIPoints(dataArchive
                                                         , "Exercise 2.*");
PIPointList pointList = new PIPointList(searchResults);
```

```
' VB - Get all points starting with "Exercise 2.*"
Dim searchResults As IEnumerable(Of PIPoint) =
    PIPoint.FindPIPoints(dataArchive, _
                        "Exercise 2.*")
Dim pointList As PIPointList = New PIPointList(searchResults)
```

5. Use the bulk **CurrentValue** method on the *pointList* to retrieve the snapshots. The **CurrentValue** method in the AF SDK returns an **AFListResults** object which contains both results and any errors that may have occurred.

```
// C# - Get all of the snapshot values
AFListResults<PIPoint, AFValue> snapshotValues =
    pointList.CurrentValue();
```

```
' VB - Get all of the snapshot values with bulk CurrentValue() call
Dim snapshotValues As AFListResults(Of PIPoint, AFValue) =
    pointList.CurrentValue
```

6. Change the code that averages the “good” snapshot values:
- Change the **if** statement to check for errors by accessing the **HasErrors** property on the results.
 - The **foreach** loop should be modified to iterate through **AFValue** objects instead of **PointValue** objects.
 - Change the **IsGood()** method call on **PIValue** to use the **IsGood** property on **AFValue**.
 - Use the **AFValue.ValueAsDouble()** method to add the current snapshot to the total.

```
// C# - If there were no errors...
if (!snapshotValues.HasErrors)
{
    foreach (AFValue snapshot in snapshotValues)
    {
        if (snapshot.IsGood)
        {
            snapshotTotal += snapshot.ValueAsDouble();
            snapshotCount++;
        }
    }
}
```



```

' VB - If there were no errors...
If Not snapshotValues.HasErrors Then
    For Each snapshot As AFValue In snapshotValues
        If snapshot.IsGood Then
            snapshotTotal += snapshot.ValueAsDouble()
            snapshotCount += 1
        End If
    Next
End If

```

7. Change the code that finds the largest count of “good” values in the set of PI Point archive values from the last 24 hours.
 - a. Remove the **PISDK** namespace qualifier from the **foreach** loop to iterate the **PIPoint** objects from the AF SDK’s **OSisoft.AF.PI** namespace.
 - b. The **RecordedValues** method call is made directly against the **PIPoint** object instead of its **Data** property as it is in the PI SDK.
 - i. Change the method call to use the AF SDK’s **AFTimeRange** to object to pass the start and end time.
 - ii. Use the new **AFBoundaryType.Inside** constant to specify the inside boundary type.
 - iii. Pass *null* and *false* as the last two parameters, since we do not want to provide a filter expression. For VB, pass *Nothing* and *False* instead.
 - c. Change the inner **foreach** loop to iterate **AFValue** objects as you did in Step 6.
 - d. Change the **IsGood()** method call on **PIValue** to use the **IsGood** property on **AFValue**.

```

// C# Snippet
foreach (PIPoint point in pointList)
{
    int goodCount = 0;

    AFValues pointValues = point.RecordedValues(
        new AFTimeRange("*-24h", "*"), AFBoundaryType.Inside, null,
        false);

    // For each value in this point's set...
    foreach (AFValue pointValue in pointValues)
    {
        // Count the 'good' ones
        if (pointValue.IsGood)
            goodCount++;
    }
    // Keep track of the highest 'good' count on a single point
    if (goodCount > maxGoodCount)
        maxGoodCount = goodCount;
}

```

VB Snippet continues next page

Exercise 2

```
' VB Snippet.
For Each point As PIPoint In pointList
    Dim goodCount As Integer = 0

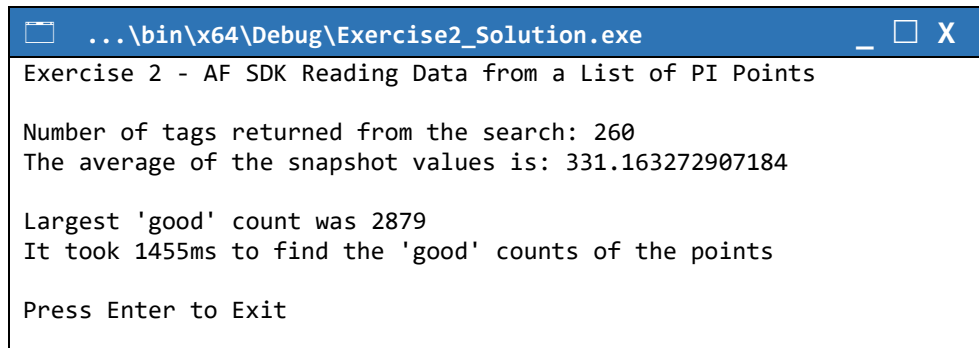
    Dim pointValues As AFValues = _
        point.RecordedValues(New AFTimeRange("*-24h", "*"), _
            AFBoundaryType.Inside, Nothing, False)

    ' For each value in this point's set...
    For Each pointValue As AFValue In pointValues
        ' Count the 'good' ones
        If pointValue.IsGood Then
            goodCount += 1
        End If

    Next

    ' Keep track of the highest 'good' count on a single point
    If (goodCount > maxGoodCount) Then
        maxGoodCount = goodCount
    End If
Next
```

8. Compile the application and run it. The application should produce similar results to the first run at the beginning of the exercise, except the AF SDK version will be faster.



```
...\\bin\\x64\\Debug\\Exercise2_Solution.exe
Exercise 2 - AF SDK Reading Data from a List of PI Points

Number of tags returned from the search: 260
The average of the snapshot values is: 331.163272907184

Largest 'good' count was 2879
It took 1455ms to find the 'good' counts of the points

Press Enter to Exit
```

4.3 Exercise 3 – Writing Data in Bulk

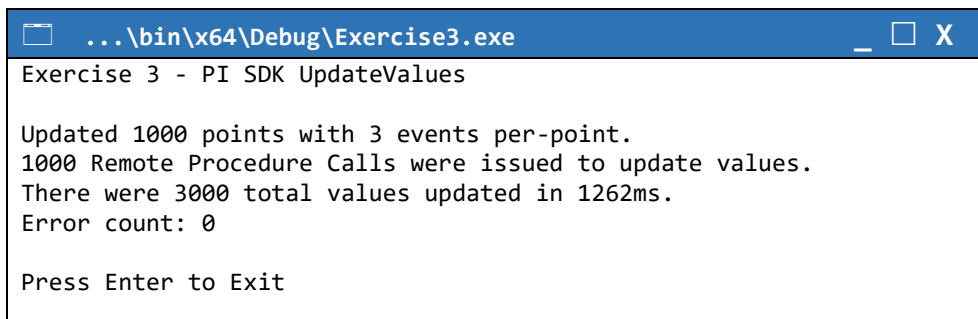
In this exercise, you will learn how to more efficiently write data, by making bulk calls to the AF SDK. Many performance issues can strictly be tied to making multiple serial calls to the PI Data Archive. Better performance is realized with fewer RPC's across the network.

This exercise is divided into two parts. In the first part you will utilize PI SDK to write multiple events to a set of PI Points, one point at a time. In the second part you will utilize AF SDK to write the same number of events to the same set of points, in bulk. The TODO and hints will be found in the second part.

1. Switch the default project to **Exercise3** via the **Set as StartUp Project** context menu option in Solution Explorer.
2. The first block of code in the template retrieves multiple PI Points from PI SDK, where the points match the name pattern "Exercise 3.*".

```
// Find some points
PISDK.PointList foundPoints = piServer.GetPoints("tag = 'Exercise 3.*'");
```

3. The next block of code in the template generates multiple values for each point, where the values are randomly generated and the timestamp difference between one event to the next is 1 minute. This is done inside a loop over each **PIPoint** that was found.
4. While still inside the loop, the next step would be to update values for the current **PIPoint**. Tracking of errors, if any, is also performed on a per **PIPoint** basis.
5. Run this **Exercise3** program and the console output should look like the figure below. Note the total elapsed time for the writes shown by the output.



```
Exercise 3 - PI SDK UpdateValues

Updated 1000 points with 3 events per-point.
1000 Remote Procedure Calls were issued to update values.
There were 3000 total values updated in 1262ms.
Error count: 0

Press Enter to Exit
```

6. The next part of the exercise will show how efficiently making a bulk write call through AF SDK for multiple points.

Exercise 3

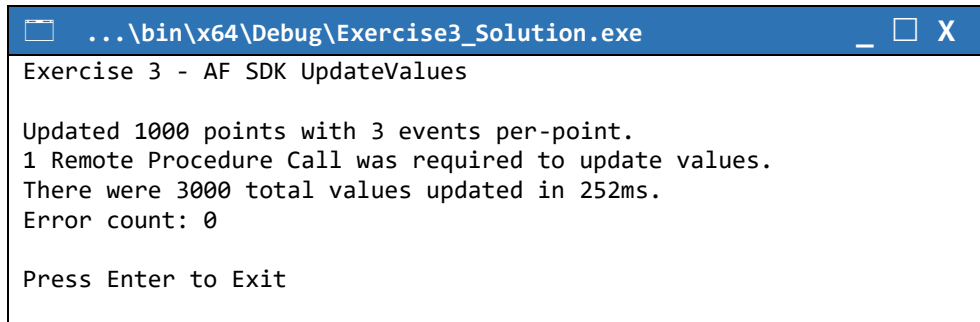
7. The following is the second part of the exercise. In the same solution, switch the default project to **Exercise3_AFSDK** via the **Set as StartUp Project** context menu option in Solution Explorer.
8. The first block of code in the template retrieves multiple PI Points from AF SDK. The **PIPoint.FindPIPoints** call has several overloads for finding points. Change this code where indicated with the “TODO” comment to find all points matching the name pattern “Exercise 3.*”. The result of a **FindPIPoints** call is an **IEnumerable** collection. Enumerating over the collection will retrieve points from the server in pages for you. Because of this, you should be careful not to enumerate the raw return more than once.

```
// Find some points
IEnumerable<PIPoint> foundPoints = PIPoint.FindPIPoints(piServer,
    "Exercise 3.*");
PIPointList pointlist = new PIPointList(foundPoints);
```

9. The next block of code in the template generates multiple values for each point, where each value has a reference to the corresponding point object. Similar to the first part of the exercise the values are randomly generated and the timestamp difference between one event to the next is 1 minute. The only difference here is an **AFValues** list is created, where each **AFValue** is associated with its corresponding **PIPoint**.
10. The following block of timed code will be used to demonstrate the advantage of making bulk calls. Replace the “TODO” comment in this block with a line of code to write multiple values for all of the points in one call.

```
AFErrors<AFValue> results = dataArchive.UpdateValues(values,
    AFUpdateOption.Replace);
```

11. Run this **Exercise3_AFSDK** program and validate the output Source code for the completed solution can be found under **Exercise3_Solution** project. The console output should look similar to the figure below. Note the total elapsed time and compare it with that from the first part of the exercise with PI SDK. This completes exercise 3.



```
... \bin\x64\Debug\Exercise3_Solution.exe
Exercise 3 - AF SDK UpdateValues

Updated 1000 points with 3 events per-point.
1 Remote Procedure Call was required to update values.
There were 3000 total values updated in 252ms.
Error count: 0

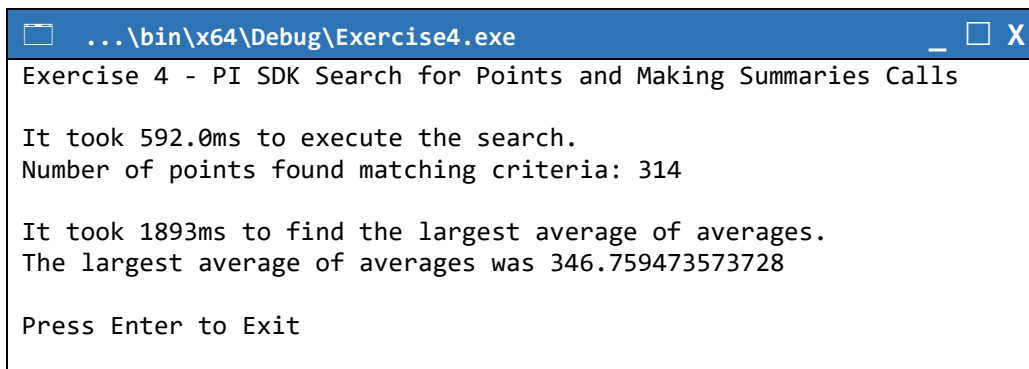
Press Enter to Exit
```


4.4 Exercise 4 – Execute Searches for PI Points and Retrieve Summary Data

In this exercise, you will convert an existing PI SDK program to use the AF SDK as you did in previous exercises. This application searches for PI Points with specific attribute values and tag naming pattern. It then queries the points for hourly averages for the last 24 hours and calculates the “average of averages.” It prints out the largest average of averages at the end.

Before you modify any code, you may be interested in a quick performance test. Refer to the section labeled “AF SDK versus PI SDK Performance Tests” to see how to do this.

1. Make sure **Exercise4** is set as the current project. The AF SDK has already been referenced in your project and the appropriate *using/Imports* statements have been added for your convenience.
2. Compile the application and run it. The output should look like the figure below



```
...\\bin\\x64\\Debug\\Exercise4.exe
Exercise 4 - PI SDK Search for Points and Making Summaries Calls

It took 592.0ms to execute the search.
Number of points found matching criteria: 314

It took 1893ms to find the largest average of averages.
The largest average of averages was 346.759473573728

Press Enter to Exit
```

3. Remove the **GetPointsSQL** call and build a list of **query** containing the same search criteria:
 - a. Create a **query** string to return only points that:
 - Have a **Tag** name containing “Exercise 4.*”.
 - Have a **PointSource** of “R”.
 - Have a **Location4** code of “1”.
 - b. Construct a **PIPointList** by passing the results from **FindPIPoints** using the query.

```
// C# snippet
string query = "PointSource='R' Location4='1' Tag='Exercise 4.*'";
PIPointList pointList = new PIPointList(
    PIPoint.FindPIPoints(dataArchive, query, false) );
```

Exercise 4

```
' VB snippet
Dim query As String = "PointSource:='R' Location4:='1' Tag:='Exercise 4.*'"
Dim pointList As PIPointList = New PIPointList( _
    PIPoint.FindPIPoints(dataArchive, query, False) )
```

4. Create a bulk call to retrieve the summaries for each PI Point in the list.
 - a. Create a **PIPaginationConfiguration** object to specify how the PI Data Archive should page the results back to the client. In this exercise we ask the data archive to send back results for 100 tags per page. Paging is abstracted from developers; however, it can impact the performance of the bulk call.
 - b. Make the **Summaries** call on the *pointList* instead of each individual point:
 - i. Create a new **AFTimeRange** to pass the start and end time for the query.
 - ii. Create a new **AFTimeSpan** to specify the summary duration.
 - iii. Specify **AFSummaryTypes.Average** for the summary type. If you wanted additional summary types to be returned you would **OR** "|" them together.
 - iv. Use the same **AFCalculationBasis** of **TimeWeightedContinuous** as the previous example used.
 - v. Automatically calculate the timestamp of the summaries by specifying **AFTimestampCalculation.Auto**.
 - vi. Pass in the **PIPaginationConfiguration** object created earlier.

```
// C# - Get a 1 hour average for the last 24 hours
PIPaginationConfiguration pagingConfig = new PIPaginationConfiguration(
    PIPageType.TagCount, 100);

IEnumerable<IDictionary<AFSummaryTypes, AFValues>> summariesResults =
    pointList.Summaries(
        new AFTimeRange("*-24h", "*"),
        new AFTimeSpan(hours: 1.0),
        AFSummaryTypes.Average,
        AFCalculationBasis.TimeWeightedContinuous,
        AFTimestampCalculation.Auto,
        pagingConfig);
```

```
' VB - Get a 1 hour average for the last 24 hours
Dim pagingConfig As PIPaginationConfiguration = _
    New PIPaginationConfiguration(PIPageType.TagCount, 100)

Dim summariesResults As IEnumerable(Of IDictionary(Of AFSummaryTypes,
    AFValues)) =
    pointList.Summaries(New AFTimeRange("*-24h", "*"),
        New AFTimeSpan(hours:=1),
        AFSummaryTypes.Average,
        AFCalculationBasis.TimeWeightedContinuous,
```



```
AFTimestampCalculation.Auto,  
pagingConfig)
```

5. Modify the **foreach** loops to work with the bulk method.
 - a. Change the outer **foreach** loop to iterate through the results from the bulk **Summaries** method which is an enumerable of dictionaries keyed on **AFSummaryTypes**.
 - b. Change the inner **foreach** loop to iterate through each **AFValue** rather than the each **PValue**.
 - c. Change the **IsGood()** method call on **PValue** to use the **IsGood** property on **AFValue**.

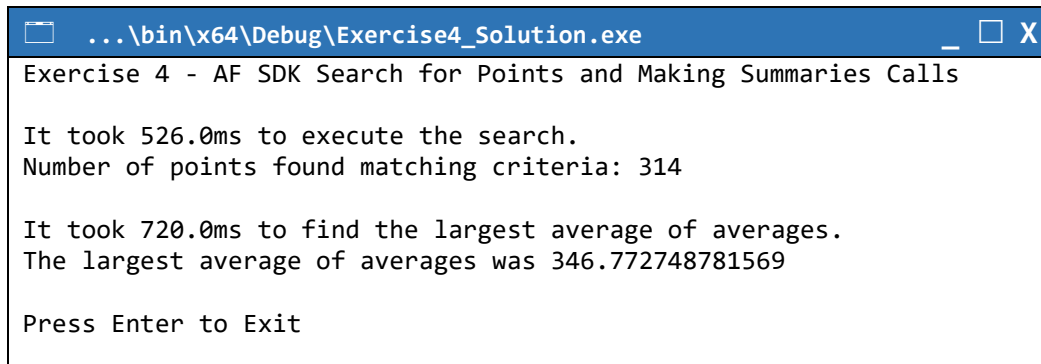
```
// C# snippet  
foreach (IDictionary<AFSummaryTypes, AFValues> summariesResult in summariesResults)  
{  
    // Find the average of averages by finding the average of hourly averages  
    double averageOfAverages = 0.0;  
    int count = 0;  
  
    foreach (AFValue average in summariesResult[AFSummaryTypes.Average])  
    {  
        if (average.IsGood)  
        {  
            averageOfAverages += average.ValueAsDouble();  
            count++;  
        }  
    }  
  
    // Calculate the average of averages  
    averageOfAverages /= count;  
  
    // Keep track of the largest one  
    if (averageOfAverages > largestAverageOfAverages)  
        largestAverageOfAverages = averageOfAverage  
}
```

```
' VB snippet  
For Each summariesResult As IDictionary(Of AFSummaryTypes, AFValues) In  
    summariesResults  
    ' Find the average of averages by finding the average of hourly averages  
    Dim averageOfAverages As Double = 0  
    Dim count As Integer = 0  
  
    For Each average As AFValue In summariesResult(AFSummaryTypes.Average)  
        If average.IsGood Then  
            averageOfAverages += average.ValueAsDouble()  
            count += 1  
        End If  
    Next  
  
    ' Calculate the average of averages  
    averageOfAverages /= count
```

Exercise 4

```
' Keep track of the largest one
If (averageOfAverages > largestAverageOfAverages) Then
    largestAverageOfAverages = averageOfAverages
End If
Next
```

6. Compile the application and run it. The application should produce similar results to the first run at the beginning of the exercise.



The screenshot shows a Windows command prompt window titled "...\\bin\\x64\\Debug\\Exercise4_Solution.exe". The window contains the following text:

```
Exercise 4 - AF SDK Search for Points and Making Summaries Calls

It took 526.0ms to execute the search.
Number of points found matching criteria: 314

It took 720.0ms to find the largest average of averages.
The largest average of averages was 346.772748781569

Press Enter to Exit
```

Appendix 1 – Bonus Applications

There are 2 bonus applications included. They are not lab exercises, nor do they have a PI SDK challenge. Because of this, each bonus application uses AF SDK only. The lab exercises use typical data calls – that is, Rich Data Access (RDA) methods to fetch tag data from the PI Data Archive. The bonus applications do not employ RDA but do show ways to extract metadata from the PI Data Archive or from a given tag.

Tag Creation Bonus with AF SDK

Tags were created for exercises 2 – 4 many days before the lab. These could have easily been created using PI Builder and a spreadsheet. However, since this is a lab on tag-based application development, it makes perfect sense that the tags are created using program code. Thus, you can see different methods and techniques that may be used to create tags with your own applications.

To see the code, go to the Visual Studio project named “**TagCreation_AFSDK_Bonus**”. Note if you tried to run this application on the lab VM, it most likely would throw an exception because the tags already exist.

If you were to first delete all tags prefaced with:

Exercise 2.*

Exercise 3.*

Exercise 4.*

Then you could create them once again without generating an exception.

PI Data Archive Metadata Bonus

The other bonus application, **PIServer_Bonus_Applet**, can be run against any PI Data Archive to display some common information such as is it a PI Collective or a stand-alone server, what is the time and time zone for the server, how many tags are on the server, or information about the point sources and classes on the server.

This helps extend your learning beyond working with mainly with tags as this applet works mainly with the PI Data Archive and various objects or properties belonging to it.

Appendix 1

Sample Console Output

PI Data Archive: PISRV01
Stand-alone server.

Host: PISRV01
Aliases (Count of 1):
localhost

Version : 3.4.425.1434
Time Zone : (UTC-08:00) Pacific Time (US & Canada)

Point Count: 1587

Point Sources (Count of 7):
9
L
Lab
PIBatch-InternalUse-1
PICampaign-InternalUse-1
PITransferRecords-InternalUse-1
R

Point Classes (Count of 2):
base
classic

StateSets (Count of 4):
SYSTEM, Count: 319
BatchAct, Count: 2
Phases, Count: 8
Modes, Count: 5

Peek into subset of SYSTEM state set (code 300-305):
Code: 300, Text: Scan Timeout
Code: 301, Text: No_Sample
Code: 302, Text: Arc Off-line
Code: 303, Text: ISU Saw No Data
Code: 304, Text: ?304
Code: 305, Text: Good

Point Attributes for SINUSOID:
descriptor = (String) 12 Hour Sine Wave
exdesc = (String)
typicalvalue = (Single) 50
engunits = (String)

```

zero = (Single) 0
span = (Single) 100
pointtype = (PIPointType) Float32
pointsource = (String) R
scan = (SByte) 1
excmin = (UInt16) 0
excmax = (UInt32) 600
excdev = (Single) 1
shutdown = (SByte) 1
archiving = (SByte) 1
compressing = (SByte) 1
step = (SByte) 0
compmin = (UInt16) 0
compmax = (UInt32) 28800
compdev = (Single) 2
creationdate = (DateTime) 12/6/2018 12:53:47 PM LocalTime
creator = (String) piadmin
changedate = (DateTime) 12/6/2018 12:53:47 PM LocalTime

```

. . . some text omitted for brevity . . .

```

dataowner = (String) piadmin
datagroup = (String) piadmins
dataaccess = (String) o:rw g:rw w:r
datasecurity = (String) piadmin: A(r,w) | piadmins: A(r,w) | PIWorld: A(r)
pointid = (UInt32) 1
recno = (UInt32) 1
future = (Byte) 0
ptclassname = (String) classic
ptclassid = (UInt32) 2
ptclassrev = (UInt32) 1
tag = (String) SINUSOID
sourcetag = (String)
digitalset = (String)
compdevpercent = (Single) 2
excdevpercent = (Single) 1

```

Press Enter to Exit

Appendix 2 – C# Cheat Sheet for VB.NET

Most of the .NET examples for many of the labs or PI Square postings are in C#. This lab is a rare exception that offers both. In other venues, occasionally there is a VB.NET post, but the VB developers at PI Square will agree that there are not enough VB examples. It would be impractical to wait for every C# example to be translated into VB.NET because that will likely not happen. Instead, you may consider yourself to be proactive and try to translate the code yourself.

Towards that end, the following link highlights some important distinctions between C# and VB.NET that can help you more easily perform the translations on your own.

CSharp Cheat Sheet for VB.NET Developers

<https://pisquare.osisoft.com/community/developers-club/blog/2018/03/19/csharp-cheat-sheet-for-vbnet-developers>

Topics covered:

- Logical Operators
 - C# **var** keyword for implicitly typed variables
 - Conditional Operator (or ? :)
 - Null-Coalescing Operator or ??
 - Read-only Auto-properties, or a get-only property
 - Expression-bodied members
 - Remainder or Modulus Operator
 - Integer versus Floating Point division
 - Using lambdas
 - Optional Parenthesis with 0 parameters
 - Characters to integers, and vice versa
-



Have an idea how to
improve our products?
**OSIsoft wants to hear
from you!**

<https://feedback.osisoft.com/>





Save the Date!

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

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

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

