AF SDK 2.9

Getting Started Guide

OSIsoft, LLC
1600 Alvarado Street
San Leandro, CA 94577 USA
Tel: (01) 510-297-5800
Fax: (01) 510-357-8136
Web: http://www.osisoft.com

AF SDK 2.9 Getting Started Guide

Version: 2.9

Published: 03 August 2017

Contents

Introduction

AF SDK (Asset Framework software development kit) is a .NET-based library that provides access to data stored in the PI System. Using AF SDK you can access:

- Hierarchically structured assets and their properties and relationships
- Time-series data from PI Data Archive and other sources
- Event frames that record and contextualize process events
- PI Data Archive, PI points, and point data

The information in this Getting Started Guide covers basic usage of AF SDK. AF SDK is one of the available developer technologies that can help you maximize the utility of your PI System.

It is recommended that you have a background in .NET application development using C# and a familiarity with the PI System; however, familiarity with AF SDK is not required.

PI developers should also consider becoming familiar with PI SQL Framework, PI Web API, or PI Builder before contemplating development with AF SDK. While AF SDK is powerful and is required in many cases, using it effectively in a complex application is easier if you already have an understanding of PI SQL or PI Web API.

When developing applications for the PI System, you have several options from which to choose, including any of the family of products that make up PI SQL Framework, PI Web API, or AF SDK. However, for best performance from your application, AF SDK has the greatest potential of all of the products in the PI System Access suite.

AF SDK provides a comprehensive, Windows-based programmatic interface to the PI System. It provides an object-oriented approach to interacting with AF structures and data, as well as the PI Data Archive directly.

There are many reasons you might choose to develop software using AF SDK:

- You can quickly build out complex PI AF hierarchies with AF SDK. Many OSIsoft partners prefer to build AF hierarchies this way, rather than use the PI AF Builder plugin to Microsoft Excel.
- AF SDK is well suited to developing middleware applications which are situated between the PI System and other systems. AF SDK gives you flexibility about how to integrate other systems with the PI System, and allows you to do so in the most resource efficient way possible. It is also possible to automate element generation and synchronization between PI AF and other systems.
- You have the ability to create rich and customized client applications with AF SDK. Most client applications that make up the PI Visualization Suite were developed using AF SDK and its predecessors.

AF SDK is designed for use with Microsoft .NET languages such as Visual Basic .NET, C#, and Managed C++. The computer you use for software development and the computer that will eventually run applications developed with AF SDK must have Microsoft .NET Framework 4.0 installed. Note that Microsoft .NET is installed automatically by the PI System Explorer installation program if it is not already installed on your computer.

About OSIsoft developer technologies

OSIsoft developer technologies PI Developer Technologies (also called PI System Access) is a family of products designed to support the implementation of custom applications that work with the PI System. PI Developer Technologies also helps you integrate PI System data with other applications and business systems such as Microsoft Office or SQL Server, enterprise resource planning (ERP) systems, reporting and analytics platforms, web portals, geospatial, and maintenance systems, to name just a few examples.

The PI System Access suite covers a wide range of use cases in various environments, programming languages, operating systems, and infrastructures. The different technologies in the PI System Access family are:

- **AF SDK**

  Provides comprehensive, high-performance, Windows-based .NET programmatic access to the PI System. AF SDK:

  - Has the best performance of all PI development technologies.
  - Has more available methods and options than any other technology.
  - Is limited to Windows operating systems running .NET Framework.

- **PI Web API**

  Enables operating system and device-independent programmatic access to the PI System through a REST API, and is the recommended cross-platform offering for any operating system that is able to use a modern web browser such as Google Chrome, Mozilla Firefox, or Microsoft Edge. PI Web API:

  - Is not dependent on a specific operating system or programming language.
  - Has limited methods and options available compared with AF SDK.

- **PI SQL Framework**

  Makes the data on PI Server available for querying as if it was on a relational database. PI SQL Framework:

  - Is compatible with other systems that use Structured Query Language (SQL).
  - Has limited methods and options available compared with AF SDK (for example, setting security on assets and creating units of measure).
  - Is limited to read-only PI AF access (but can create and write to tags on your PI System).

There are more ways to access the PI System programmatically; however, the above three methods describe the vast majority of use cases.

You might consider using the tools for Microsoft PowerShell. See the OSIsoft Tech Support page PI Powershell (https://techsupport.osisoft.com/Documentation/PI-Powershell/title.html). PowerShell cmdlets help with administration of PI Data Archive and PI Asset Framework servers, and can be used in a variety of ways to create scripts for commonly needed functionality or for bulk system management operations.

Documentation for these products (and others) can be found in: Live Library (https://livelibrary.osisoft.com/).

AF SDK versus PI SDK

OSIsoft recommends using AF SDK for all new software development projects. If you have experience coding with the older PI SDK, you should be aware of the following differences:

- AF SDK cannot be used with ModuleDB, PIModules, and so on.
- AF SDK cannot be used with PI message logs (for this, you should use PowerShell instead) .

Applications that do not reference any of the items in the above list will execute faster if they are written using AF SDK.

In addition, AF SDK uses managed .NET code, whereas PI SDK uses un-managed code that relies upon Microsoft COM (Component Object Module), which might be awkward to use for beginning developers.

Another difference is that AF SDK uses the `AFValue` object while PI SDK uses a `PIValue` object. While a `PIValue` is composed of a `Value` and `Timestamp` property, an AFValue includes a `UOM` (unit of measure) property.

Virtually everything you could do with PI SDK (with only limited exceptions), you can do with AF SDK.

Where to find help

You can obtain AF SDK help online at the OSIsoft Tech Support page AF SDK RFeference (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/1a02af4c-1bec-4804-a9ef-3c7300f5e2fc.htm).

You can also view a help file that is installed when you install AF SDK. Navigate to the %`PIHOME%\help` directory and double-click the `AFSDK.chm` file.

You can also display help directly from PI System Explorer by selecting **Help** > **AF SDK Reference**.

PI Developers Club within PI Square is also available for obtaining help online. PI Developers Club is a one-stop shop for all PI System development needs. See PI Developers Club (https://pisquare.osisoft.com/community/developers-club).

Configuration

The information in this topic and the following topics describes how to configure your development environment to create applications using AF SDK.

Note:
The rest of the topics in this section provide information about how to configure your system so you can run the programming exercises in this guide. These steps are required only if you are configuring your own environment and are not necessary if you are using the Virtual Learning Environment (VLE).

There are two ways to configure an environment for developing applications. The easiest way is to simply sign up with OSIsoft Learning or PI Developers Club space within PI Square and create a Virtual Learning Environment (VLE). A VLE is a virtual computer that is configured with the appropriate versions of PI Data Archive, PI AF, Visual Studio, .NET, and AF SDK. Other than creating the VLE, no additional configuration is required. A VLE is available for this Getting Started Guide at a nominal cost. See the OSIsoft Learning page Virtual Learning Environment Overview (https://learning.osisoft.com/Labs/LabInformation) for details.

The other way to configure an environment for developing applications is to use an existing PI Data Archive and PI AF database at your company or organization. To run the examples in this Getting Started Guide you must have the following:

- An operational PI Data Archive and PI AF system, preferably a system used for testing or QA work. Because this Getting Started Guide creates and edits tags and asset elements, you should not use a production system.
- AF SDK version 2.9.1. You can download the latest version from OSIsoft Tech Support (https://techsupport.osisoft.com/Downloads/All-Downloads/Developer-Technologies/All-Products).
- .NET version 4.5. See the OSIsoft Tech Support page .NET 4.5 differences (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/cc26e52e-b3fc-4b0b-9ff5-0526a8b4c8cc.htm) for more information.
- Microsoft Visual Studio 2015 or Microsoft Visual Studio Community edition (available free). Visual Studio Express 2017 is also available free from Microsoft.

Create the database for your PI system

The steps in this section create the database and sample data you need for the exercises in this guide.

The steps in this section are necessary only if you are creating your own environment and are not required if you are using a Virtual Learning Environment.

Procedure

1. Clone a local copy of the GitHub repository.

   The code for the exercises is located in GitHub:
   `https://github.com/osisoft/AF-SDK-Getting-Started-Guide`

2. Start PI System Explorer and click **File** > **Database**

The Select Database window displays.

3. Click **New Database**.

   The Database Properties window displays.

4. In the **Name** field, enter the name of the database to create; in this case, `Green Power Company`.

   Also provide a brief description, such as: `Intro to AF SDK exercise database`.

5. Click **OK** when finished, then click **Close**.

6. Click **File** > **Import from File**

7. In the **File** text box, browse to and select `AF Database - Green Power Company.xml`.

```
AF Database - Green Power Company.xml
```

8. Click **OK** when finished.

   The data for the exercises is created and stored in the Green Power Company database.

Create an application structure

Microsoft Visual Studio creates the outline of a program when you create a new project. Follow the steps in this section to create a program outline. Subsequent topics show how to customize the application and augment it with AF SDK information.

Procedure

1. If necessary, start Visual Studio.
2. Click **File** > **New** > **Project**

   The New Project window displays.

3. Select **Console Application**, then enter a name for the project in the **Name** field.

   For this example, enter `PI_Connect` as the project name. Visual Studio creates the project and presents a program outline that includes the **Main** function. You augment this outline with your own code.

Set the target framework

The steps in this section set or verify that the target framework is set correctly in Visual Studio.

Procedure

1. In Solution Explorer window, right click the program name. A tab opens with properties for your application.
2. If necessary, click **Application**.
3. In the **Assembly name** list, select **`.NET Framework 4.5.2`**.
4. Close the tab.

Add AF SDK references to your project

To make Visual Studio compile your code using AF SDK references, you use Reference Manager. The extensions section of Reference Manager lists assemblies that OSIsoft has developed to extend the .NET Framework.

Procedure

1. In Visual Studio, select **Project** > **Add reference**.

   The Reference Manager window displays.

2. Click the **Extensions** node.

   A list of custom assemblies displays.

3. Scroll the list until you see items from OSIsoft.

   You can also enter a search term such as `OSI` in the **Search Assemblies** box.

4. Select the check box next to **OSIsoft AFSDK version 4.0.0.0**.

   Be sure to select version OSIsoft AFSDK and not OSIsoft.PISDK. Also be sure to select version 4 and not version 2.

5. Click **OK**.

   The Reference Manager window closes and saves your changes.

   The references you added using Reference Manager apply to the entire project, which means that the referenced dynamic link libraries (DLLs or library files) are available to the project. However, the referenced DLLs are quite large and contain lots of functionality that your program might not use.

Specify namespaces

You must specify which namespaces you want to work with for a given DLL. For a given application, you must add the appropriate `using` statements at the top of the program to indicate the namespaces that the application will be using frequently.

Note:
When you run an application created with AF SDK, the application dynamically calls AF SDK dynamic link libraries (DLLs); however, the application also requires some of the services that are installed with AF SDK in addition to the DLLs. Therefore, you must install PI System Explorer on any computer on which an AF SDK application is run.

Procedure

1. Using the Visual Studio editor, insert the following lines at the beginning of the program:

```
using OSIsoft.AF;
using OSIsoft.AF.Asset;
using OSIsoft.AF.UnitsOfMeasure;
```

   For later class files or projects that you work on, some other popular "usings" are:

```
using OSIsoft.AF.Time;
using OSIsoft.AF.Data;
using OSIsoft.AF.EventFrame;
```

The using statements you add depend on which of the many namespaces in a given DLL you want to have available to the program you are working on.

Programming exercises

In each of the lessons that follow, you will be tasked with creating methods that are representative of the types of work you perform using AF SDK. Each lesson introduces the tasks to be performed, followed by hints and tips about the lesson. In addition, working code is provided for each of the lesson exercises.

Lessons 1 through 3 can be completed even if you have only a basic understanding of PI system and AF, while lessons 4 and 5 require more in-depth knowledge of PI and how to perform certain operations. Complete source code is included for every lesson and programing exercise, and explanatory text contains code snippets you can use to augment your understanding of the material.

Lesson 1: Connect to PI Data Archive or PI AF database

After completing this lesson you will be able to:

- Describe the structure of a PI AF database
- Create a simple console application
- Use AF SDK to create an application to connect to a PI AF server or PI Data Archive
- Use the `PISystems` class to find a list of known PI AF servers, and the `PIServers` class to find the list of PI Data Archive servers
- Describe the AF SDK internal cache
- Connect to PI AF server and PI Data Archive collectives

Help for methods and data types used in this lesson

The following links contain information that describes important objects used in this lesson. Refer to these pages on the OSIsoft Support site when writing code for the exercises:

- AFDatabase Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_AFDatabase.htm)
- AFElementTemplate Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/Html/T_OSIsoft_AF_Asset_AFElementTemplate.htm)
- AFDatabase.ElementTemplates.FilterBy() Method (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSIsoft_AF_AFSDKExtension_FilterBy.htm)
- AFNamedCollectionList Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_AFNamedCollectionList_1.htm)
- AFAttributeTemplate Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_Asset_AFAttributeTemplate.htm)
- UOMClass Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_UnitsOfMeasure_UOMClass.htm)
- AFEnumerationSets Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_Asset_AFEnumerationSets.htm)
- AFCategories Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_AFCategories.htm)

Introduction

For this lesson, you will access a PI AF database named Green Power Company, which has already been configured on your local PI AF server. You will be asked to print to the console the names and properties of various PI AF objects. You should familiarize yourself with the Green Power Company PI AF database using PI System Explorer before starting the exercises.

AF SDK basics

AF SDK presents a hierarchical model of objects and collections that represent underlying PI System features and concepts. Using AF SDK, you use the `PISystem` object as an abstraction for your PI System.

The `PISystem` object is the top-level object used to represent a logical PI AF server. Each `PISystem` can have one or more separate `AFDatabase` objects. Each `AFDatabase` object may contain any number of child AFElement objects, which are sometimes referred to as assets.

Each `AFElement` object may have child `AFElements` or child `AFAttributes`. `AFAttributes` of an `AFElement` may represent values from a data stream, table, or descriptive values (either static or dynamically changing). The `PISystem` object is ideal for organizing assets that may be grouped in hierarchies and classified using templates created to describe similar objects across the database.

Note that the UOM (unit of measure) database is attached to a `PISystem` object. In addition, only one UOM database exists in a PI System.

Objects and collections

Many PI AF objects have associated collections. For example, each `AFElement` may contain any number of child elements. The group of child elements is represented as an `AFElements` object. The reference between an object and its child collection is used to build the hierarchical structure of PI AF. Note that AF SDK employs a consistent naming convention for collections; for example, a collection of `AFElement` objects is named `AFElements`.

The programmatic object hierarchy is very similar to the visual hierarchy displayed in PI System Explorer. Therefore, understanding how objects are related within PI System Explorer greatly facilitates learning AF SDK.

Namespaces

AF SDK is organized into several namespaces. The most common namespaces and their basic functions are listed here. A block diagram of the PISystem Hierarchy (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/EB961F37-282A-43D2-8F8C-F19CE07D9FA8.htm) is available in help.

- **OSIsoft.AF**

  The `AF` namespace contains the main classes used by all the other namespaces within AF SDK. It provides base class implementations and methods for connecting to PI AF servers. The primary classes are `PISystem` and `AFDatabase`. Each PI System is composed of any number of distinct databases. For more information see the OSIsoft Tech Support page PISystem hierarchy (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/EB961F37-282A-43D2-8F8C-F19CE07D9FA8.htm).

- **OSIsoft.AF.Asset**

  The `Asset` namespace provides a set of classes for representing assets within an organization. It allows the creation of hierarchies of assets and their attributes. Additionally, it provides features for dealing with common requirements such as remote data access, unit-of-measure conversion, and defining and enforcing asset definitions. Templates for assets ensure consistency among attributes. In the AF namespace, you will find `AFElement`, `AFAttribute`, `AFElementTemplates`, and so on, along with their associated collections (such as `AFElements`). In addition, `AFValue`, used to represent a time-series event, is located in the `AF` namespace. For more information see the OSIsoft Tech Support page Asset namespace (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/N_OSIsoft_AF_Asset.htm).

- **OSIsoft.AF.Data**

  The `Data` namespace provides classes for obtaining rich data access (RDA) from assets within an organization. Both historical and real-time data access are provided. Data access across asset types allow for simpler architecture as well as helpful functionality such as unit-of-measure conversions. For more information see the OSIsoft Tech Support page Data namespace (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/N_OSIsoft_AF_Data.htm).

- **OSIsoft.AF.PI**

  The `PI` namespace provides classes that are used to manage connections and access information about PI Data Archive. It provides direct access to PI Data Archive and can be used to find or edit PI points and read or write data directly to PI Data Archive. See the AF SDK online reference for more information. For more information see the OSIsoft Tech Support page AF.PI namespace (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/N_OSIsoft_AF_PI.htm).

- **OSIsoft.AF.Time**

  The `AF.Time` namespace provides classes for performing time operations with the PI System. It provides methods for converting time strings including abbreviations in the PI time syntax (such as "*-5m") to an `AFTime` and converting .NET `DateTime` structures to and from AF time structures. Time, time range, and time interval classes also facilitate accurate representation across time zones and across daylight transitions. For more information see the OSIsoft Tech Support page AF.Time namespace (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/N_OSIsoft_AF_Time.htm).

- **OSIsoft.AF.EventFrame**

  The `EventFrame` namespace provides classes for reading and writing PI AF Event Frames. These objects can be tied to PI AF elements and represent events that occur over a time period, such as equipment downtime or abnormal behavior. For more information see the OSIsoft Tech Support page EventFrame namespace (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/N_OSIsoft_AF_EventFrame.htm).

Namespaces are brought into scope in your application by adding the appropriate `using` directives at the top of the class file. For example:

```
using OSIsoft.AF;
using OSIsoft.AF.Asset;
using OSIsoft.AF.Data;
```

Connect to a PI Server or AF Database

You can create a list of known AF servers by first initializing an instance of the `PISystems` collection. The list of all PI Data Archive servers that are known to the client is represented as `PIServers`. The code below shows how to obtain an object called *My AF Server* that represents the PI AF server and an object called *My PI Data Archive* that represents PI Data Archive.

```
PISystems piSystems = new PISystems();
PISystem assetServer = piSystems["My AF Server"];

PIServers piServers = new PIServers();
PIServer piServer = piServers["My PI Data Archive"];
```

After you obtain a reference to the `PISystem`, you can refer to its collection of `AFDatabase` objects by using the `Databases` property of `piSystem`, as shown in the following code:

```
AFDatabase database = assetServer.Databases["Meters"];
```

To return a specific `AFDatabase`, you pass a name to the `AFDatabases` indexer.

You can obtain additional references from the `AFDatabase` to other collections such as root elements, categories, templates, and tables. To obtain a specific object from a collection, use the collection's indexer and pass in a name, as shown in the following code.

```
AFElements rootElements = database.Elements;
AFElement element1 = rootElements["Meter1"];
AFCategories attributeCategories = database.AttributeCategories;
AFElementTemplates elementTemplates = database.ElementTemplates;
AFTables tables = database.Tables;
```

Notice that the object hierarchy corresponds very closely to the hierarchy you see in PI System Explorer. Remember to use PI System Explorer as a guide for illustrating object relationships within AF SDK.

Note that there is no explicit call to a `Connect`() method. AF SDK connects to the PI AF database automatically as required. This automatic connection is called an implicit connection. To make an explicit connection, you can call `Connect`() on a `PISystem` instance. Implicit and explicit connections are discussed in more detail in the AF SDK Online Reference (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/a1616e71-fb2e-40b1-9f14-299b2d9b269c.htm).

Example

```
static void PrintRootElements(AFDatabase database)
{
  Console.WriteLine("Print Root Elements: {0}", database.Elements.Count);
  foreach (AFElement element in database.Elements)
  {
    Console.WriteLine("  {0}", element.Name);
  }

  Console.WriteLine();
}
```

Lesson one exercises

For each of the following exercises, add logic to each method to accomplish the given task. Add the methods to your console application and test each one by writing to the console.

- **Exercise 1**

  Create a method with the definition and arguments shown below. Incorporate the method into your console application and print the name of each `AFElementTemplate`. For each element template, also print the name of each `AFCategory` in its Categories collection.

  ```
  void PrintElementTemplates( AFDatabase database )
  ```

  When the method is run, it should produce the following output:

  ```
  Print Element Templates
  Name: City; Categories:
  Name: MeterAdvanced; Categories: Shows Status;
  Name: MeterBasic; Categories: Measures Energy;
  ```

- **Exercise 2**

  Create a method with the definition and arguments shown below. Incorporate the method into your console application and print the name of each `AFAttributeTemplate` for the passed-in element template. For each attribute template, also print the name of its data reference plug-in.

  ```
  void PrintAttributeTemplates(AFDatabase database, string elemTempName)
  ```

  When the method is run, it should produce the following output:

  ```
  Print Attribute Templates for Element Template: MeterAdvanced
  Name: Status, DRPlugin: PI Point
  ```

- **Exercise 3**

  Create a method with the definition and arguments shown below. Incorporate the method into your console application and print the name of each unit of measure (UOM) and its abbreviation under the `UOMClass` `Energy`.

  ```
  void PrintEnergyUOMs( PISystem system )
  ```

  When the method is run, it should produce the following output:

  ```
  Print Energy UOMs
  UOM: gigawatt hour, Abbreviation: GWh
  UOM: megawatt hour, Abbreviation: MWh
  UOM: watt hour, Abbreviation: Wh
  UOM: joule, Abbreviation: J
  UOM: British thermal unit, Abbreviation: Btu
  UOM: calorie, Abbreviation: cal
  UOM: gigajoule, Abbreviation: GJ
  UOM: kilojoule, Abbreviation: kJ
  UOM: kilowatt hour, Abbreviation: kWh
  UOM: megajoule, Abbreviation: MJ
  UOM: watt second, Abbreviation: Ws
  UOM: kilocalorie, Abbreviation: kcal
  UOM: million calorie, Abbreviation: MMcal
  UOM: million British thermal unit, Abbreviation: MM Btu
  ```

- **Exercise 4**

  Create a method with the definition and arguments shown below. Incorporate the method into your console application and print the name of each `AFEnumerationSet` object. For each set, print its list of states.

  ```
  void PrintEnumerationSets( AFDatabase database )
  ```

  When the method is run, it should produce the following output:

  ```
  Print Enumeration Sets
  Building Type
  0 - A
  1 - B

  Meter Status
  0 - Good
  1 - Bad
  ```

- **Exercise 5**

  Create a method with the definition and arguments shown below. Incorporate the method into your console application and print the name of each element category and attribute category.

```
void PrintCategories( AFDatabase database )
```

When the method is run, it should produce the following output:

```
Print Categories
Element Categories
Measures Energy
Shows Status

Attribute Categories
Building Info
Location
Time-Series Data
```

Lesson 1 hints and tips

AF SDK maintains an internal cache for each PI AF identity. For example, if two users have exactly the same access to the PI AF database, then they would share the same cache. The cache is initialized when the first connection to the database is made. Internally, AF SDK allows connections to the PI AF server and PI Data Archive to drop after periods of inactivity and automatically reestablishes connection when activity is resumed. The user cache is preserved between periods of inactivity and can be reused after reconnecting. Calling `Disconnect()` explicitly, however, clears the user cache.

When using a `PISystem` or `PIServer` instance, OSIsoft does not recommend that you manually incorporate your own reconnect logic using `Disconnect()` and `Connect()`; doing so forces the cache to re-initialize upon reconnecting. In most of your projects with AF SDK, OSIsoft recommends using implicit connections and having AF SDK handle reconnecting.

Lesson 1 exercise solutions

This section contains sample code that solves each of the exercises in this lesson.

- **Exercise 1**

```
static void PrintElementTemplates(AFDatabase database)
{
  Console.WriteLine("Print Element Templates");
  AFNamedCollectionList<AFElementTemplate> elemTemplates =
    database.ElementTemplates.FilterBy(typeof(AFElement));
  foreach (AFElementTemplate elemTemp in elemTemplates)
  {
    Console.WriteLine("Name: {0}; Categories: {1}", elemTemp.Name,
      elemTemp.CategoriesString);
  }

  Console.WriteLine();
}
```

- **Exercise 2**

```
static void PrintAttributeTemplates(AFDatabase database, string elemTempName)
{
  Console.WriteLine("Print Attribute Templates for Element Template:
    {0}", elemTempName);
  AFElementTemplate elemTemp = database.ElementTemplates[elemTempName];
  foreach (AFAttributeTemplate attrTemp in elemTemp.AttributeTemplates)
  {
```

```
    string drName = attrTemp.DataReferencePlugIn ==
      null ? "None" : attrTemp.DataReferencePlugIn.Name;
    Console.WriteLine("Name: {0}, DRPlugin: {1}", attrTemp.Name, drName);
  }
  Console.WriteLine();
}
```

- **Exercise 3**

```
static void PrintEnergyUOMs(PISystem system)
{
  Console.WriteLine("Print Energy UOMs");
  UOMClass uomClass = system.UOMDatabase.UOMClasses["Energy"];
  foreach (UOM uom in uomClass.UOMs)
  {
    Console.WriteLine("UOM: {0}, Abbreviation: {1}", uom.Name,
      uom.Abbreviation);
  }
  Console.WriteLine();
}
```

Note that the UOMDatabase belongs to PISystem and is not exclusive to an AFDatabase.

- **Exercise 4**

```
static void PrintEnumerationSets(AFDatabase database)
{
  Console.WriteLine("Print Enumeration Sets");
  AFEnumerationSets enumSets = database.EnumerationSets;
  foreach (AFEnumerationSet enumSet in enumSets)
  {
    Console.WriteLine(enumSet.Name);
    foreach (AFEnumerationValue state in enumSet)
    {
      int stateValue = state.Value;
      string stateName = state.Name;
      Console.WriteLine("{0} - {1}", stateValue, stateName);
    }

    Console.WriteLine();
  }
}
```

- **Exercise 5**

```
static void PrintCategories(AFDatabase database)
{
  Console.WriteLine("Print Categories");
  AFCategories elemCategories = database.ElementCategories;
  AFCategories attrCategories = database.AttributeCategories;

  Console.WriteLine("Element Categories");
  foreach (AFCategory category in elemCategories)
  {
    Console.WriteLine(category.Name);
  }

  Console.WriteLine();
  Console.WriteLine("Attribute Categories");
  foreach (AFCategory category in attrCategories)
  {
    Console.WriteLine(category.Name);
  }
```

```
  Console.WriteLine();
}
```

Lesson 2: Searching for assets

After completing this lesson you will be able to:

- Use AF SDK to search for assets (elements and their attributes).
- Experiment with the different search criteria that can be used.
- Evaluate how to handle search results.

Help for methods and data types used in this lesson

- AFElement Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_Asset_AFElement.htm)
- AFElementSearch Constructor (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSIsoft_AF_Search_AFElementSearch__ctor_1.htm)
- AFElementSearch.FindElements() Method (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSIsoft_AF_Search_AFElementSearch_FindElements.htm)
- AFAttribute.Categories Property (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/P_OSIsoft_AF_Asset_AFAttribute_Categories.htm)

Introduction

OSIsoft recommends that you reduce the search results to a minimum on the server, not on the client. In other words, you should call AF SDK methods that allow the server to process the search results instead of writing custom search logic in the client code.

Starting with PI AF 2016 (2.8), search query methods are available that process query strings instead of methods for each type of search. For the exercises in this section, use the new `AFSearch` methods such as the `AFElementSearch` class.

For this lesson, imagine you are a software developer for a power company. Your assignment is to develop a client application that allows users to search for electric meters (PI AF Elements) and attributes. When the application is complete, users should be able to search for meters by name, element template, element category, and attribute values. The user should also be able to search for meter attributes by name, element template, and attribute category.

Use the PI AF database named Green Power Company, which is located on your PI AF server. You will be asked to write several methods that search for elements and attributes and print the search results to the console. Before starting, it might help to familiarize yourself with the Green Power Company database using PI System Explorer.

Example

The first example problem is shown here along with its solution. For this first problem, you should implement the following methods inside `Program2.cs` and run the application with the arguments provided for each of the following methods in `Main()`.

```
static void FindMetersByName(AFDatabase database, string elementNameFilter)
  {
    Console.WriteLine("Find Meters by Name: {0}", elementNameFilter);

    // Default search is as an element name string mask.
    string querystring = string.Format("{0}", elementNameFilter);
    AFElementSearch elementquery = new AFElementSearch(database,
```

```
      "ElementSearch", querystring);
  foreach (AFElement element in elementquery.FindElements())
  {
    Console.WriteLine("Element: {0}, Template: {1}, Categories: {2}",
       element.Name,
       element.Template.Name,
       element.CategoriesString);
  }
  Console.WriteLine();
```

Lesson 2 exercises

For each of the following exercises, your task is to add logic to each method. Add the methods to your console application and test each one by writing output to the console.

- **Exercise 1**

  Find all meters that were created from the provided element template and derived templates. Print the names of each found meter in a list and their element template names. Count the number of derived templates that were found.

  ```
  void FindMetersByTemplate( AFDatabase database, string templateName)
  ```

  When the method is run, it should product the following output:

  ```
  Find Meters by Name: Meter00*
  Element: Meter001, Template: MeterBasic, Categories: Measures Energy;
  Element: Meter002, Template: MeterBasic, Categories: Measures Energy;
  Element: Meter003, Template: MeterBasic, Categories: Measures Energy;
  Element: Meter004, Template: MeterBasic, Categories: Measures Energy;
  Element: Meter005, Template: MeterBasic, Categories: Measures Energy;
  Element: Meter006, Template: MeterBasic, Categories: Measures Energy;
  Element: Meter007, Template: MeterBasic, Categories: Measures Energy;
  Element: Meter008, Template: MeterBasic, Categories: Measures Energy;
  Element: Meter009, Template: MeterAdvanced, Categories: Measures Energy;Shows
  Status;
  ```

- **Exercise 2**

  Find all meters supplied by the provided substation. Print the names of each of the meters that were found.

  ```
  void FindMetersBySubstation( AFDatabase database, string substationLocation)
  ```

  When the method is run, it should product the following output:

  ```
  Find Meters by Template: MeterBasic
  Element: Meter001, Template: MeterBasic
  Element: Meter002, Template: MeterBasic
  Element: Meter003, Template: MeterBasic
  Element: Meter004, Template: MeterBasic
  Element: Meter005, Template: MeterBasic
  Element: Meter006, Template: MeterBasic
  Element: Meter007, Template: MeterBasic
  Element: Meter008, Template: MeterBasic
  Element: Meter009, Template: MeterAdvanced
  Element: Meter010, Template: MeterAdvanced
  Element: Meter011, Template: MeterAdvanced
  Element: Meter012, Template: MeterAdvanced
     Found 4 derived templates
  ```

- **Exercise 3**

  Find all meters with energy usage greater than the supplied limit. Print the names of each meter that was found.

  ```
  void FindMetersAboveUsage( AFDatabase database, double limit)
  ```

  When the method is run, it should product the following output:

  ```
  Find Meters above Usage: 300
  Meter003, Meter005, Meter006, Meter008, Meter009, Meter012
  ```

- **Exercise 4**

  Find all attributes belonging to the `Building Info` attribute category that are part of the provided element template. Print the number of attributes found.

  ```
  void FindBuildingInfo( AFDatabase database, string templateName)
  ```

  When the method is run, it should product the following output:

  ```
  Find Building Info: MeterAdvanced
  Found 8 attributes.
  ```

Lesson 2 hints and tips

A general pattern for searching is shown here:

```
using (AFElementSearch elementQuery = new AFElementSearch(database,
       "nameOfSearch", queryString))
{
    elementQuery.CacheTimeout = TimeSpan.FromMinutes(5);
    foreach (AFElement element in elementQuery.FindElements())
    {
        ProcessElement(element);
    }
}
```

Lesson 2 exercise solutions

- **Exercise 1: Find meters by template**

```
static void FindMetersByTemplate(AFDatabase database, string templateName)
{
  Console.WriteLine("Find Meters by Template: {0}", templateName);

  using (AFElementSearch elementQuery =
    new AFElementSearch(database, "TemplateSearch",
        string.Format("template:\"{0}\"", templateName)))
    {
      elementQuery.CacheTimeout = TimeSpan.FromMinutes(5);
      int countDerived = 0;
      foreach (AFElement element in elementQuery.FindElements())
      {
         Console.WriteLine("Element: {0}, Template: {1}", element.Name,
           element.Template.Name);
         if (element.Template.Name != templateName)
            countDerived++;
      }

      Console.WriteLine("   Found {0} derived templates", countDerived);
      Console.WriteLine();
```

```
    }
}
```

- **Exercise 2: Find meters by substation**

```
static void FindMetersBySubstation(AFDatabase database, string
substationLocation)
{
  Console.WriteLine("Find Meters by Substation: {0}", substationLocation);

  string templateName = "MeterBasic";
  string attributeName = "Substation";
  using (AFElementSearch elementQuery =
    new AFElementSearch(database, "AttributeValueEQSearch",
    string.Format("template:\"{0}\" \"|{1}\":\"{2}\"",
    templateName, attributeName, substationLocation)))

  {
    elementQuery.CacheTimeout = TimeSpan.FromMinutes(5);
    int countNames = 0;
    foreach (AFElement element in elementQuery.FindElements())
    {
      Console.Write("{0}{1}", countNames++ == 0 ? string.Empty : ", ",
        element.Name);
    }

    Console.WriteLine();
  }
}
```

- **Exercise 3: Find meters with above-average usage:**

```
static void FindMetersAboveUsage(AFDatabase database, double val)
{
  Console.WriteLine("Find Meters above Usage: {0}", val);

  string templateName = "MeterBasic";
  string attributeName = "Energy Usage";
  AFElementSearch elementquery = new AFElementSearch(database,
    "AttributeValueGTSearch",
  string.Format("template:\"{0}\" \"|{1}\":>{2}", templateName,
    attributeName, val));

  int countNames = 0;
  foreach (AFElement element in elementquery.FindElements())
  {
     Console.Write("{0}{1}", countNames++ ==
       0 ? string.Empty : ", ", element.Name);
  }

  Console.WriteLine();
}
```

- **Exercise 4: Find building information**

```
static void FindBuildingInfo(AFDatabase database, string templateName)
{
   Console.WriteLine("Find Building Info: {0}", templateName);

   AFElementTemplate elemTemp = database.ElementTemplates[templateName];
   AFCategory buildingInfoCat = database.AttributeCategories["Building Info"];

   AFSearchToken token = new AFSearchToken(AFSearchFilter.Template,
     AFSearchOperator.Equal, elemTemp.GetPath());
```

```
  AFElementSearch search =
    new AFElementSearch(database, "search", new[] { token });

  IEnumerable<AFElement> foundElements = search.FindElements();
  AFNamedCollectionList<AFAttribute> foundAttributes =
    new AFNamedCollectionList<AFAttribute>();


  foreach (AFElement foundElem in foundElements)
  {
    foreach (AFAttribute attr in foundElem.Attributes)
    {
      if (attr.Categories.Contains(buildingInfoCat))
      {
         foundAttributes.Add(attr);
      }
    }
  }

  Console.WriteLine("Found {0} attributes.", foundAttributes.Count);
  Console.WriteLine();
}
```

Lesson 3: Reading and writing data

After completing this lesson you will be able to:

- Use AF SDK to read data from PI AF
- Use AF SDK to write data to PI AF
- Use bulk calls to retrieve data within a specified time range for many attributes
- Use the `AFValue` object to store time-series information

Help for methods and data types used in this lesson

- AFValue Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_Asset_AFValue.htm)
- PIPagingConfiguration() Constructor (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSIsoft_AF_PI_PIPagingConfiguration__ctor.htm)
- AFAttribute ListClass (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_Asset_AFAttributeList.htm)

Bulk calls

To retrieve data within a specified time range for many attributes, it is time-consuming to issue a data call for each attribute, because many round trips over the network to the PI Data Archive are required. In networked systems, latency delays can be costly.

You can reduce network latency (the number of network round trips) by using list calls in AF SDK. First, you create a list of attributes to retrieve and store them in an `AFAttributeList`. You then access the `Data` property on the `AFAttributeList`, which exposes data retrieval methods for the entire attribute list. Now, you can make a single call to the server to retrieve data for all of the attributes, as shown in the following code:

```
AFAttributeList attrList = AFAttribute.FindElementAttributes(
  // argument list omitted for brevity
  );
  // Bulk call to get value at specified time for all found attributes

  IList<AFValue> vals = attrList.Data.RecordedValue(time);
```

You might be wondering why the above example used `AFAttributeList` instead of `AFAttributes`. The reason in that `AFAttributes` returns a list of attributes for an element, whereas `AFAttributeList` returns a list of unrelated `AFAttributes`. For more information, see AFAttributes (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_Asset_AFAttributes.htm) and AFAttributeList (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_Asset_AFAttributeList.htm).

The `AFValue` object contains two important properties:

- `AFValue.Timestamp`: An `AFTime` object that represents the event's timestamp
- `AFValue.Value`: A `.NET` object representing the event's value that can be cast to the appropriate type

`AFTime` is similar to a .NET `DateTime` object; however, `AFTime` objects can be constructed using PI Time syntax (for example: "t" or "*-10m").

Example

To read time-series data, you first obtain a reference to the `AFAttribute`, and then access its `Data` property. The data property is an `AFData` object that exposes methods to retrieve time-series data. In the example here, all of the archived values of an attribute for the last 10 minutes are retrieved.

```
AFTime startTime = new AFTime("*-10m");
AFTime endTime = new AFTime("*");
AFTimeRange timeRange = new AFTimeRange(startTime, endTime);
AFValues vals = attr.Data.RecordedValues(
  timeRange: timeRange,
  boundaryType: AFBoundaryType.Inside,
  desiredUOM: null,
  filterExpression: null,
  includeFilteredValues: false);
```

Lesson 3 exercises

Your assignment is to develop a client application that allows users to read data from and write data to the PI Data Archive.

You will use the PI AF database Green Power Company located on your local PI AF server. You will be asked to write several methods to retrieve historical data from the PI Data Archive and print the results to the console. You will also implement methods to write data. Before starting the exercises, use PI System Explorer to familiarize yourself with the PI AF database.

For this lesson, implement the following methods inside `Program3.cs` and run the application with the arguments provided for each of the following methods in `Main()`.

- **Exercise 1**

  Create a method to retrieve interpolated values for the **`Energy Usage`** attribute for the specified meter element. The **`startTime`** and **`endTime`** should be in PI Time syntax. Print the following information for each returned `AFValue`: Timestamp (Local time), Value in kilowatt hours.

  ```
  void PrintInterpolated(AFDatabase database, string meterName,
       string startTime, string endTime, TimeSpan timeSpan)
  ```

  When the method is run, it should product the following output:

  ```
  Print Interpolated Values - Meter: Meter001, Start: *-30s, End: *
  Timestamp (Local): 7/6/2017 4:14:29 PM, Value: 271.21 kWh
  Timestamp (Local): 7/6/2017 4:14:39 PM, Value: 270.92 kWh
  Timestamp (Local): 7/6/2017 4:14:49 PM, Value: 270.92 kWh
  Timestamp (Local): 7/6/2017 4:14:59 PM, Value: 270.92 kWh
  ```

- **Exercise 2**

  Create a method to print the hourly time-weighted average for the Energy Usage attribute for the specified meter element. The **`startTime`** and **`endTime`** should be in PI Time syntax with time range larger than one hour. Print the following information for each returned AFValue: Timestamp (Local time), Value in kilowatt hours.

  ```
  void PrintHourlyAverage(AFDatabase database, string meterName,
       string startTime, string endTime)
  ```

  When the method is run, it should product the following output:

```
Print Hourly Average - Meter: Meter001, Start: y, End: t
Timestamp (Local): 2017-07-05 00h, Value: 219.71 kWh
Timestamp (Local): 2017-07-05 01h, Value: 269.12 kWh
Timestamp (Local): 2017-07-05 02h, Value: 287.00 kWh
Timestamp (Local): 2017-07-05 03h, Value: 321.93 kWh
Timestamp (Local): 2017-07-05 04h, Value: 304.84 kWh
Timestamp (Local): 2017-07-05 05h, Value: 319.51 kWh
Timestamp (Local): 2017-07-05 06h, Value: 312.74 kWh
Timestamp (Local): 2017-07-05 07h, Value: 307.56 kWh
Timestamp (Local): 2017-07-05 08h, Value: 326.19 kWh
Timestamp (Local): 2017-07-05 09h, Value: 283.09 kWh
Timestamp (Local): 2017-07-05 10h, Value: 325.61 kWh
Timestamp (Local): 2017-07-05 11h, Value: 319.55 kWh
Timestamp (Local): 2017-07-05 12h, Value: 308.70 kWh
Timestamp (Local): 2017-07-05 13h, Value: 241.24 kWh
Timestamp (Local): 2017-07-05 14h, Value: 291.32 kWh
Timestamp (Local): 2017-07-05 15h, Value: 300.86 kWh
Timestamp (Local): 2017-07-05 16h, Value: 314.04 kWh
Timestamp (Local): 2017-07-05 17h, Value: 297.21 kWh
Timestamp (Local): 2017-07-05 18h, Value: 343.67 kWh
Timestamp (Local): 2017-07-05 19h, Value: 281.79 kWh
Timestamp (Local): 2017-07-05 20h, Value: 273.63 kWh
Timestamp (Local): 2017-07-05 21h, Value: 289.00 kWh
Timestamp (Local): 2017-07-05 22h, Value: 282.71 kWh
Timestamp (Local): 2017-07-05 23h, Value: 310.37 kWh
```

- **Exercise 3**

  Create a method to retrieve the archived event at the given timestamp for the Energy Usage attributes for all meters. The method should use the `AFAttributeList` object. Print the following information for each meter: Meter name, Timestamp (Local time), Value in kilowatt hour.

  ```
  void PrintEnergyUsageAtTime(AFDatabase database, string timeStamp)
  ```

  When the method is run, it should product the following output:

  ```
  Print Energy Usage at Time: t+10h
  Meter: Meter001, Timestamp (Local): 2017-07-06 10h, Value: 240.79 kWh
  Meter: Meter002, Timestamp (Local): 2017-07-06 10h, Value: 320.99 kWh
  Meter: Meter003, Timestamp (Local): 2017-07-06 10h, Value: 146.76 kWh
  Meter: Meter004, Timestamp (Local): 2017-07-06 10h, Value: 247.14 kWh
  Meter: Meter005, Timestamp (Local): 2017-07-06 10h, Value: 300.47 kWh
  Meter: Meter006, Timestamp (Local): 2017-07-06 10h, Value: 343.78 kWh
  Meter: Meter007, Timestamp (Local): 2017-07-06 10h, Value: 300.69 kWh
  Meter: Meter008, Timestamp (Local): 2017-07-06 10h, Value: 368.51 kWh
  Meter: Meter009, Timestamp (Local): 2017-07-06 10h, Value: 255.79 kWh
  Meter: Meter010, Timestamp (Local): 2017-07-06 10h, Value: 387.83 kWh
  Meter: Meter011, Timestamp (Local): 2017-07-06 10h, Value: 254.49 kWh
  Meter: Meter012, Timestamp (Local): 2017-07-06 10h, Value: 391.02 kWh
  ```

- **Exercise 4**

  Create a method that retrieves the daily averages of the Energy Usage attribute for all meters. The **`startTime`** and **`endTime`** should be in PI Time syntax. The method should use the `AFAttributeList` object. Print the following information for each meter: Meter name, Timestamp (Local time), Avg. Value in kilowatt hours.

  ```
  void PrintDailyAverageEnergyUsage(AFDatabase database, string startTime,
       string endTime)
  ```

  When the method is run, it should product the following output:

```
Print Daily Energy Usage - Start: t-7d, End: t
Averages for Meter: Meter001
Timestamp (Local): 2017-06-29, Avg. Value: 292.41 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 296.29 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 288.61 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 289.56 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 305.59 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 300.72 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 297.14 kWh

Averages for Meter: Meter002
Timestamp (Local): 2017-06-29, Avg. Value: 290.59 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 301.80 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 313.02 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 296.27 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 296.36 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 294.80 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 290.91 kWh

Averages for Meter: Meter003
Timestamp (Local): 2017-06-29, Avg. Value: 306.56 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 296.34 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 297.13 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 303.96 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 296.18 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 296.94 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 294.30 kWh

Averages for Meter: Meter004
Timestamp (Local): 2017-06-29, Avg. Value: 290.80 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 303.45 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 300.44 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 295.50 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 300.97 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 302.34 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 290.79 kWh

Averages for Meter: Meter005
Timestamp (Local): 2017-06-29, Avg. Value: 295.32 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 296.99 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 306.82 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 305.22 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 304.02 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 301.04 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 306.42 kWh

Averages for Meter: Meter006
Timestamp (Local): 2017-06-29, Avg. Value: 304.98 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 300.31 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 314.78 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 300.96 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 314.68 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 304.36 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 292.20 kWh

Averages for Meter: Meter007
Timestamp (Local): 2017-06-29, Avg. Value: 299.20 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 300.51 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 296.11 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 311.83 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 307.76 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 300.82 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 303.01 kWh

Averages for Meter: Meter008
```

```
Timestamp (Local): 2017-06-29, Avg. Value: 298.25 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 304.60 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 294.96 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 308.05 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 299.51 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 295.44 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 304.02 kWh

Averages for Meter: Meter009
Timestamp (Local): 2017-06-29, Avg. Value: 291.10 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 300.96 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 293.08 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 297.69 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 297.84 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 298.78 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 309.21 kWh

Averages for Meter: Meter010
Timestamp (Local): 2017-06-29, Avg. Value: 311.84 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 295.10 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 296.29 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 298.10 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 298.69 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 298.91 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 299.91 kWh

Averages for Meter: Meter011
Timestamp (Local): 2017-06-29, Avg. Value: 297.88 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 303.83 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 299.66 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 303.63 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 297.02 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 299.82 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 306.44 kWh

Averages for Meter: Meter012
Timestamp (Local): 2017-06-29, Avg. Value: 291.31 kWh
Timestamp (Local): 2017-06-30, Avg. Value: 301.20 kWh
Timestamp (Local): 2017-07-01, Avg. Value: 299.30 kWh
Timestamp (Local): 2017-07-02, Avg. Value: 295.43 kWh
Timestamp (Local): 2017-07-03, Avg. Value: 299.08 kWh
Timestamp (Local): 2017-07-04, Avg. Value: 306.22 kWh
Timestamp (Local): 2017-07-05, Avg. Value: 300.89 kWh
```

- **Exercise 5**

  Occasionally, the Energy Usage meter data is incorrect. The values for one meter actually belong to another meter and vice versa. Therefore, implement the method from Exercise 4 which takes the name of two meters and start and end time in PI Time syntax. The method should swap the Energy Usage attribute values between the two meters within the given time range. Try to implement this with only two `UpdateValues` calls. What are some tradeoffs when limiting the number of remote calls in this case? How would you ensure there is no data loss?

  ```
  void SwapValues(AFDatabase database, string meter1, string meter2,
       string startTime, string endTime)
  ```

  When the method is run, it should product the following output:

  ```
  Swap values for meters: Meter001, Meter002 between y and y+1h
  ```

Lesson 3 hints and tips

For exercises 4 and 5, use the `GetAttributes()` helper function to obtain an **`AFAttributeList`**, then use `AFAttributeList.Data` to make the bulk calls.

For exercise 4, do not use `RecordedValues()` and then compute the average in your code. Instead, use the `Summaries()` method to allow PI Data Archive to compute the average. Then, return only the average to the client.

For exercise 5, the existing archived values must be removed before inserting new values. Use the `AFListData.UpdateValues` in conjunction with `AFUpdateOption.Remove`.

Lesson 3 exercise solutions

- **Exercise 1, print interpolated values**

```
static void PrintInterpolated(AFDatabase database, string meterName,
  string startTime, string endTime, TimeSpan timeSpan)
{
  Console.WriteLine(string.Format("Print Interpolated Values - Meter: {0},
    Start: {1}, End: {2}", meterName, startTime, endTime));

  AFAttribute attr =
    AFAttribute.FindAttribute(@"\Meters\" + meterName + @"|Energy Usage",
      database);

  AFTime start = new AFTime(startTime);
  AFTime end = new AFTime(endTime);
  AFTimeRange timeRange = new AFTimeRange(start, end);

  AFTimeSpan interval = new AFTimeSpan(timeSpan);

  AFValues vals = attr.Data.InterpolatedValues(
      timeRange: timeRange,
      interval: interval,
      desiredUOM: null,
      filterExpression: null,
      includeFilteredValues: false);

  foreach (AFValue val in vals)
  {
     Console.WriteLine("Timestamp (Local): {0},
       Value: {1:0.00} {2}", val.Timestamp.LocalTime, val.Value,
       val?.UOM.Abbreviation);
  }
  Console.WriteLine();
}
```

- **Exercise 2, print hourly average**

```
static void PrintHourlyAverage(AFDatabase database, string meterName,
  string startTime, string endTime)
{
  Console.WriteLine(string.Format
    ("Print Hourly Average - Meter: {0}, Start: {1}, End: {2}", meterName,
      startTime, endTime));

  AFAttribute attr =
    AFAttribute.FindAttribute(@"\Meters\" + meterName + @"|Energy Usage",
      database);
```

```
  AFTime start = new AFTime(startTime);
  AFTime end = new AFTime(endTime);
  AFTimeRange timeRange = new AFTimeRange(start, end);

  IDictionary<AFSummaryTypes, AFValues> vals = attr.Data.Summaries(
      timeRange: timeRange,
      summaryDuration: new AFTimeSpan(TimeSpan.FromHours(1)),
      summaryType: AFSummaryTypes.Average,
      calcBasis: AFCalculationBasis.TimeWeighted,
      timeType: AFTimestampCalculation.EarliestTime);


  foreach (AFValue val in vals[AFSummaryTypes.Average])
  {
    Console.WriteLine("Timestamp (Local):
      {0:yyyy-MM-dd HH\\h}, Value: {1:0.00} {2}", val.Timestamp.LocalTime,
        val.Value,val?.UOM.Abbreviation);
  }

  Console.WriteLine();
}
```

- **Exercise 3, print energy usage**

```
static void PrintEnergyUsageAtTime(AFDatabase database, string timeStamp)
{
  Console.WriteLine("Print Energy Usage at Time: {0}", timeStamp);

  AFAttributeList attrList = GetAttributes(database, "MeterBasic",
    "Energy Usage");

  AFTime time = new AFTime(timeStamp);

  IList<AFValue> vals = attrList.Data.RecordedValue(time);
  foreach (AFValue val in vals)
  {
    Console.WriteLine("Meter: {0},
      Timestamp (Local): {1:yyyy-MM-dd HH\\h}, Value: {2:0.00} {3}",
      val.Attribute.Element.Name,val.Timestamp.LocalTime,
        val.Value,val?.UOM.Abbreviation);
  }
  Console.WriteLine();
}
```

- **Exercise 4, print daily average energy usage**

```
static void PrintDailyAverageEnergyUsage(AFDatabase database,
  string startTime, string endTime)
{
  Console.WriteLine(string.Format("Print Daily Energy Usage - Start: {0},
    End: {1}", startTime, endTime));

  AFAttributeList attrList = GetAttributes(database,
    "MeterBasic", "Energy Usage");

  AFTime start = new AFTime(startTime);
  AFTime end = new AFTime(endTime);
  AFTimeRange timeRange = new AFTimeRange(start, end);

  // Ask for 100 PI Points at a time
  PIPagingConfiguration pagingConfig =
    new PIPagingConfiguration(PIPageType.TagCount, 100);

  IEnumerable<IDictionary<AFSummaryTypes, AFValues>> summaries =
    attrList.Data.Summaries(
  timeRange: timeRange,
```

```
  summaryDuration: new AFTimeSpan(TimeSpan.FromDays(1)),
  summaryTypes: AFSummaryTypes.Average,
  calculationBasis: AFCalculationBasis.TimeWeighted,
  timeType: AFTimestampCalculation.EarliestTime,
  pagingConfig: pagingConfig);

  // Loop through attributes
  foreach (IDictionary<AFSummaryTypes, AFValues> dict in summaries)
  {
    AFValues values = dict[AFSummaryTypes.Average];
    Console.WriteLine("Averages for Meter: {0}",
      values.Attribute.Element.Name);

    // Loop through values per attribute
    foreach (AFValue val in dict[AFSummaryTypes.Average])
    {
      Console.WriteLine("Timestamp (Local): {0:yyyy-MM-dd},
      Avg. Value: {1:0.00} {2}",
      val.Timestamp.LocalTime,val.Value,val?.UOM.Abbreviation);
    }
    Console.WriteLine();

  }
  Console.WriteLine();
}
```

- **Exercise 5, swap values**

```
static void SwapValues(AFDatabase database, string meter1, string meter2,
  string startTime, string endTime)
{
  Console.WriteLine
    (string.Format("Swap values for meters: {0}, {1} between {2} and {3}",
     meter1, meter2, startTime, endTime));

  // NOTE: This method does not ensure that there is no data loss
  // if there is failure.
  // Persist the data first in case you need to rollback.

  AFAttribute attr1 = AFAttribute.FindAttribute(@"\Meters\" +
    meter1 + @"|Energy Usage", database);
  AFAttribute attr2 = AFAttribute.FindAttribute(@"\Meters\" +
    meter2 + @"|Energy Usage", database);

  AFTime start = new AFTime(startTime);
  AFTime end = new AFTime(endTime);
  AFTimeRange timeRange = new AFTimeRange(start, end);

  // Get values to delete for meter1
  AFValues valsToRemove1 = attr1.Data.RecordedValues(
     timeRange: timeRange,
     boundaryType: AFBoundaryType.Inside,
     desiredUOM: null,
     filterExpression: null,
     includeFilteredValues: false);

  // Get values to delete for meter2
  AFValues valsToRemove2 = attr2.Data.RecordedValues(
     timeRange: timeRange,
     boundaryType: AFBoundaryType.Inside,
     desiredUOM: null,
     filterExpression: null,
     includeFilteredValues: false);

  List<AFValue> valsToRemove = valsToRemove1.ToList();
  valsToRemove.AddRange(valsToRemove2.ToList());
```

```
  // Remove the values
  AFListData.UpdateValues(valsToRemove, AFUpdateOption.Remove);

  // Create new AFValues from the other meter and assign them to this meter
  List<AFValue> valsToAdd1 = valsToRemove2.Select(v => new AFValue(attr1,
    v.Value, v.Timestamp)).ToList();
  List<AFValue> valsToAdd2 = valsToRemove1.Select(v => new AFValue(attr2,
    v.Value, v.Timestamp)).ToList();

  List<AFValue> valsCombined = valsToAdd1;
  valsCombined.AddRange(valsToAdd2);

  AFListData.UpdateValues(valsCombined, AFUpdateOption.Insert);
  Console.WriteLine();
}
```

Lesson 4: Building an AF hierarchy

After completing this lesson you will be able to:

- Use AF SDK to build a PI AF hierarchy.
- Create a PI AF database.
- Create element and attribute templates.
- Create an asset hierarchy, and link to PI Points from the attribute configuration.
- Use recommended AF SDK patterns such as checking in changes in bulk.

Help for methods and data types used in this lesson

- AFDatabase.ElementTemplate.Add() Method (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSIsoft_AF_Asset_AFElementTemplates_Add.htm)
- AFDatabase.Elements.Add() Method (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/Html/M_OSIsoft_AF_Asset_AFElements_Add_3.htm)
- AFDatabase.CheckIn() Method (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSIsoft_AF_AFDatabase_CheckIn.htm)
- AFReferenceType Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_Asset_AFReferenceType.htm)

Introduction

When attempting to add an element to a database, you should always check to see if an element with the same name already exists in the database before adding it. Attempting to add an element that already exists in the database will generate an exception.

The following code shows one way to add an element named **Meters** to the database:

```
// Attempts to retrieve the element "Meters", and adds the element after
checking
// to see if it already exists
AFElement meters = database.Elements["Meters"]
if (meters == null)
    meters = database.Elements.Add("Meters");
```

Alternatively, the previous operation could be written using a single line by using the C# coalesce operator: **??**, as shown here:

```
AFElement meters = database.Elements["Meters"] ?? database.Elements.Add("Meters")
```

Shown below is a simple example that shows how to create a new PI AF database, PI AF element, and PI AF attribute, and then check in the changes. Refer to the Example 4 exercises section for more detailed examples.

```
PISystem assetServer = new PISystems().DefaultPISystem;

AFDatabase database = assetServer.Databases.Add("MyDb");

AFElement newEl = database.Elements[“MyElement”] ??
database.Elements.Add(“MyElement”);

AFAttribute newAttr = newEl.Attributes["MyAttribute"] ??
newE1.Attributes.Add(MyAttribute");
```

```
database.CheckIn();
```

Example

The following example shows how to create an element template called `FeederTemplate`. The example creates one attribute template called `District` which contains no data reference, then creates another attribute template called `Power`. The default UOM is set to **`watt`** and the `Type` to **`Single`**. The example also sets the data reference to **`PI Point`**.

See the following links for more information:

- Configuration of data references (https://livelibrary.osisoft.com/LiveLibrary/content/en/server-v9/GUID-FBC4ED44-9448-4C17-95FE-BA55AA5CABF1)
- Units of measure in PI AF (https://livelibrary.osisoft.com/LiveLibrary/content/en/server-v9/GUID-B6ABBFA8-22EE-42E3-84F3-BBAA638EFE58)
- PI point data reference (https://livelibrary.osisoft.com/LiveLibrary/content/en/server-v9/GUID-9FA38FDC-4325-4222-BBBF-926DC1183685)

```
static void CreateElementTemplate(AFDatabase database)
{
  string templateName = "FeederTemplate";
  AFElementTemplate feederTemplate;
  if (database.ElementTemplates.Contains(templateName))
    return;
  else
    feederTemplate = database.ElementTemplates.Add(templateName);

    AFAttributeTemplate cityAttributeTemplate =
      feederTemplate.AttributeTemplates.Add("City");
    cityattributeTemplate.Type = typeof(string);

    AFAttributeTemplate power = feederTemplate.AttributeTemplates.Add("Power");
    power.Type = typeof(Single);

    power.DefaultUOM = database.assetServer.UOMDatabase.UOMs["watt"];
    power.DataReferencePlugIn =
      database.PISystem.DataReferencePlugIns["PI Point"];

    database.CheckIn();
 }
```

Lesson 4 exercises

- **Exercise 1**

  Create an element called `Feeders` directly under the database root.

- **Exercise 2**

  Create an element called `Feeder001` under `Feeders` based off of the `FeederTemplate` element template. Set the value of the `City` attribute to `London`. Set the `ConfigString` property of the `Power` attribute such that the attribute's data source is the `SINUSOID` PI Point.

- **Exercise 3**

  Create a weak reference (https://livelibrary.osisoft.com/LiveLibrary/content/en/server-v9/GUID-4318CCE5-2EBC-4239-BA04-8F0B1808BD53) between `Feeder001` and the `Feeder001` element. The result should resemble the following structure:

```
Elements
    - Configuration
    - Feeders
        - Feeder001
    - Geographical Locations
        - London
            - Feeder001
            - Meter001
            - Meter002
            - Meter003
            - Meter004
        + Montreal
        + San Francisco
    + Meters
```

Bonus exercises

Create a replica of the AF Database `Green Power Company`.

1. Create an AF Database named `Ethical Power Company`.

```
AFDatabase CreateDatabase( string servername, string databasename )
```

2. Create the element categories and attribute categories.

```
void CreateCategories( AFDatabase database )
```

3. Create the enumeration sets and their values.

```
void CreateEnumerationSets( AFDatabase database )
```

4. Create the PI AF Element Templates and their attribute templates. Set the properties and categories based on the reference database.

```
void CreateTemplates( AFDatabase database )
```

5. Create a root element called `Meters`. Create individual meter elements from the appropriate templates. For Meter001 through Meter008 use `MeterBasic`. For Meter009 through Meter012 use `MeterAdvanced`.

```
void CreateElements( AFDatabase database )
```

6. Set the attribute values of the meter as appropriate based on the reference database. Only do this for the first meter.

```
void SetAttributeValues( AFDatabase database )
```

7. Create the root element called `Geographical Locations`. Then, create three child elements called `London`, `Montreal`, and `San Francisco`, based on the `District` template.

```
void CreateDistrictElements( AFDatabase database )
```

8. Add meter elements as weak references under the `District` elements, following the reference database.

```
void CreateWeakReferences( AFDatabase database )
```

Lesson 4 hints and tips

- **Create an AFEnumerationSet**

```
AFEnumerationSet bTypeEnum = database.EnumerationSets.Add("Building Type");
bTypeEnum.Add("Residential", 0);
bTypeEnum.Add("Business", 1);
```

- **Set AFAttributeTemplate properties**

```
AFAttributeTemplate energyUsageAttrTemp =
  meterBasicTemplate.AttributeTemplates.Add("Energy Usage");
energyUsageAttrTemp.Type = typeof(Single);
energyUsageAttrTemp.Categories.Add(tsDataA);
energyUsageAttrTemp.DefaultUOM = uom;
energyUsageAttrTemp.DataReferencePlugIn =
  database.PISystem.DataReferencePlugIns["PI Point"];
energyUsageAttrTemp.ConfigString =
  @"\\%@\Configuration|PIDataArchiveName%\%Element%.%Attribute%;UOM=kWh";
```

- **Set AFAttributeTemplate type to be an enumeration set**

```
AFAttributeTemplate attrTemp = elemTemp.AttributeTemplates["Building Type"];
AFEnumerationSet enumSet = database.EnumerationSets["Building Type"];
attrTemp.TypeQualifier = enumSet;
```

- **Check in the changes**

```
if (database.IsDirty)
  database.CheckIn();
```

- **Commit changes using bulk CheckIn**

  In the exercise solution code, you perform `CheckIn` operations in bulk to improve performance. This is accomplished by calling `CheckIn` less frequently. For most situations, avoid calling `CheckIn` on every object change because each causes a call to the SQL server. When making multiple modifications, a general rule of thumb is to check In 200 objects at a time. If more control is required over the list of objects to be checked in, use the `CheckIn()` method on a PISystem instance. For more information, see PISystem.Checkin Method (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSISOFT_AF_PISYSTEM_CHECKIN_2.htm).

- **Add Overloads**

  When adding `AFElement` objects to an `AFElements` collection, several `Add` methods are available. The methods can be roughly divided into two groups:

- Methods that accept a string denoting the new `AFElement` name and return a reference to the newly created `AFElement`.

```
AFElement meters = database.Elements.Add("Meters");
```

- Methods that accept a reference to an existing `AFElement` object and return void

A common use case for the second method is when adding an existing element as a weak reference to a parent element as in the following:

```
AFReferenceType weakRefType = database.ReferenceTypes["Weak Reference"];
AFElement meters = database.Elements["Meters"];
AFElement buildingA = database.Elements["Meter Company"].Elements["Building
A"];
buildingA.Elements.Add(meters.Elements["Meter001"], weakRefType);
```

Lesson 4 exercise solutions

- **Exercise 1: Create an element named Feeders**

```
static void CreateFeedersRootElement(AFDatabase database)
{
  if (database.Elements.Contains("Feeders"))
    return;

  database.Elements.Add("Feeders");
  database.CheckIn();
}
```

- **Exercise 2: Create an element from template**

```
static void CreateFeederElements(AFDatabase database)
{
  AFElementTemplate template = database.ElementTemplates["FeederTemplate"];

  AFElement feeders = database.Elements["Feeders"];
  if (template == null || feeders == null) return;

  if (feeders.Elements.Contains("Feeder001")) return;

  AFElement feeder001 = feeders.Elements.Add("Feeder001", template);

  AFAttribute city = feeder001.Attributes["City"];
  if (city != null) city.SetValue(new AFValue("London"));

  AFAttribute power = feeder001.Attributes["Power"];
  power.ConfigString = @"%@\Configuration|PIDataArchiveName%\SINUSOID";

  if (database.IsDirty)
    database.CheckIn();
}
```

- **Exercise 3: Create a weak reference**

```
static void CreateWeakReference(AFDatabase database)
{
  AFReferenceType weakRefType = database.ReferenceTypes["Weak Reference"];

  AFElement london =
    database.Elements["Geographical Locations"].Elements["London"];
  AFElement feeder0001 = database.Elements["Feeders"].Elements["Feeder001"];
```

```
  if (london == null || feeder0001 == null) return;

  if (!london.Elements.Contains(feeder0001))
    london.Elements.Add(feeder0001, weakRefType);

  database.CheckIn();
}
```

Answers to bonus exercises

1. Create an AF Database named `Ethical Power Company`.

```
private static AFDatabase GetOrCreateDatabase(string servername, string
databasename)
{
  PISystem assetServer = new PISystems()[servername];
  if (assetServer == null)
    return null;
  AFDatabase database = assetServer.Databases[databasename];
  if (database == null)
    database = assetServer.Databases.Add(databasename);
  return database;
}
```

2. Create the element categories and attribute categories.

```
private static void CreateCategories(AFDatabase database)
{
  if (database == null) return;
  var items = new List<string>
  {
    "Measures Energy",
    "Shows Status",
    "Building Info",
    "Location",
    "Time - Series Data"
  };
  foreach (var item in items)
  {
    if (!database.ElementCategories.Contains(item))
      database.ElementCategories.Add(item);
  }
  if (database.IsDirty)
    database.CheckIn();
}
```

3. Create the enumeration sets and their values.

```
private static void CreateEnumerationSets(AFDatabase database)
{
  if (database == null) return;

  if (!database.EnumerationSets.Contains("Building Type"))
  {
     AFEnumerationSet bTypeEnum =
       database.EnumerationSets.Add("Building Type");
     bTypeEnum.Add("Residential", 0);
     bTypeEnum.Add("Business", 1);
  }
  if (!database.EnumerationSets.Contains("Meter Status"))
  {
     AFEnumerationSet mStatusEnum =
       database.EnumerationSets.Add("Meter Status");
     mStatusEnum.Add("Good", 0);
     mStatusEnum.Add("Bad", 1);
  }
```

```
  if (database.IsDirty)
    database.CheckIn();
}
```

4. Create the PI AF Element Templates and their attribute templates. Set the properties and categories based on the reference database.

   In order to modularize the code and to limit each method to performing only one task, the following solution was divided into four methods:

```
private static void CreateTemplates(AFDatabase database)
{
  if (database == null) return;
  AFElementTemplate meterBasicTemplate = CreateMeterBasicTemplate(database);
  CreateMeterAdvancedTemplate(meterBasicTemplate);
  CreateCityTemplate(database);

  if (database.IsDirty)
    database.CheckIn();
}
```

```
private static AFElementTemplate CreateMeterBasicTemplate(AFDatabase database)
{
  AFElementTemplate meterBasicTemplate =
    database.ElementTemplates["MeterBasic"];
  if (meterBasicTemplate != null)
    return meterBasicTemplate;

  UOM uom = database.PISystem.UOMDatabase.UOMs["kilowatt hour"];
  AFCategory mEnergyE = database.ElementCategories["Measures Energy"];
  AFCategory bInfoA = database.AttributeCategories["Building Info"];
  AFCategory locationA = database.AttributeCategories["Location"];
  AFCategory tsDataA = database.AttributeCategories["Time-Series Data"];
  AFEnumerationSet bTypeNum = database.EnumerationSets["Building Type"];

  meterBasicTemplate = database.ElementTemplates.Add("MeterBasic");
  meterBasicTemplate.Categories.Add(mEnergyE);

  AFAttributeTemplate substationAttrTemp =
    meterBasicTemplate.AttributeTemplates.Add("Substation");
  substationAttrTemp.Type = typeof(string);

  AFAttributeTemplate usageLimitAttrTemp =
    meterBasicTemplate.AttributeTemplates.Add("Usage Limit");
  usageLimitAttrTemp.Type = typeof(string);
  usageLimitAttrTemp.DefaultUOM = uom;

  AFAttributeTemplate buildingAttrTemp =
    meterBasicTemplate.AttributeTemplates.Add("Building");
  buildingAttrTemp.Type = typeof(string);
  buildingAttrTemp.Categories.Add(bInfoA);

  AFAttributeTemplate bTypeAttrTemp =
    meterBasicTemplate.AttributeTemplates.Add("Building Type");
  bTypeAttrTemp.TypeQualifier = bTypeNum;
  bTypeAttrTemp.Categories.Add(bInfoA);

  AFAttributeTemplate cityAttrTemp =
    meterBasicTemplate.AttributeTemplates.Add("City");
  cityAttrTemp.Type = typeof(string);
  cityAttrTemp.Categories.Add(locationA);

  AFAttributeTemplate energyUsageAttrTemp =
    meterBasicTemplate.AttributeTemplates.Add("Energy Usage");
  energyUsageAttrTemp.Type = typeof(Single);
```

```
  energyUsageAttrTemp.Categories.Add(tsDataA);
  energyUsageAttrTemp.DefaultUOM = uom;

  energyUsageAttrTemp.DataReferencePlugIn =
    database.PISystem.DataReferencePlugIns["PI Point"];
  energyUsageAttrTemp.ConfigString =
    @"\\%@\Configuration|PIDataArchiveName%\%Element%.%Attribute%;UOM=kWh";

  return meterBasicTemplate;
}
```

```
private static void CreateMeterAdvancedTemplate(AFElementTemplate
meterBasicTemplate)
{
  AFDatabase database = meterBasicTemplate.Database;
        AFElementTemplate meterAdvancedTemplate =
    database.ElementTemplates["MeterAdvanced"];
        if (meterAdvancedTemplate == null)
       database.ElementTemplates.Add("MeterAdvanced");

        AFCategory tsDataA =
    database.AttributeCategories["Time-Series Data"];
        AFEnumerationSet mStatusEnum =
    database.EnumerationSets["Meter Status"];

     meterAdvancedTemplate.BaseTemplate = meterBasicTemplate;

     AFAttributeTemplate statusAttrTemp =
    meterAdvancedTemplate.AttributeTemplates["Status"];
        if (statusAttrTemp == null)
       meterAdvancedTemplate.AttributeTemplates.Add("Status");
        statusAttrTemp.TypeQualifier = mStatusEnum;
     if (!statusAttrTemp.Categories.Contains(tsDataA))
                statusAttrTemp.Categories.Add(tsDataA);
        statusAttrTemp.DataReferencePlugIn =
    database.PISystem.DataReferencePlugIns["PI Point"];
        statusAttrTemp.ConfigString =
    @"\\%@\Configuration|PIDataArchiveName%\%Element%.%Attribute%";
}
```

```
private static void CreateCityTemplate(AFDatabase database)
{
  AFElementTemplate cityTemplate = database.ElementTemplates["City"];
  if (cityTemplate != null)
    return;
  AFAttributeTemplate cityEnergyUsageAttrTemp =
    cityTemplate.AttributeTemplates.Add("Energy Usage");
  cityEnergyUsageAttrTemp.Type = typeof(Single);
  cityEnergyUsageAttrTemp.DefaultUOM =
    database.PISystem.UOMDatabase.UOMs["kilowatt hour"];
  cityEnergyUsageAttrTemp.DataReferencePlugIn =
    database.PISystem.DataReferencePlugIns["PI Point"];
  cityEnergyUsageAttrTemp.ConfigString =
    @"\\%@\Configuration|PIDataArchiveName%\%Element%.%Attribute%";
}
```

5. Create a root element called "Meters". Create individual meter elements from the appropriate templates. Meter001-008 use MeterBasic. The rest use MeterAdvanced.

```
private static void CreateElements(AFDatabase database)
{
  if (database == null) return;
  // here we create the configuration element
  // we do a small exception creating an attribute in this method.
  AFElement configuration;
```

```
  if (!database.Elements.Contains("Configuration"))
  {
    configuration = database.Elements.Add("Configuration");
    AFAttribute name= configuration.Attributes.Add("PIDataArchiveName");
    name.SetValue(new AFValue("PISRV01"));
  }

  AFElement meters = database.Elements["Meters"];
  if (meters == null)
    meters = database.Elements.Add("Meters");

  AFElementTemplate basic = database.ElementTemplates["MeterBasic"];
  AFElementTemplate advanced = database.ElementTemplates["MeterAdvanced"];

  foreach (int i in Enumerable.Range(1, 12))
  {
    string name = "Meter" + i.ToString("D3");
    if (!meters.Elements.Contains(name))
    {
      AFElementTemplate eTemp = i <= 8 ? basic : advanced;
      AFElement e = meters.Elements.Add(name, eTemp);
    }
  }

  if (database.IsDirty)
    database.CheckIn();
}
```

6. Set the attribute values of the meter as appropriate based on the reference database. Only do this for the first meter.

```
private static void SetAttributeValues(AFDatabase database)
{
  if (database == null) return;

  AFElement meter001 = database.Elements["Meters"].Elements["Meter001"];
  meter001.Attributes["Substation"].SetValue(new AFValue("SSA-01"));
  meter001.Attributes["Usage Limit"].SetValue(new AFValue(350));
  meter001.Attributes["Building"].SetValue(new AFValue("The Shard"));

  AFEnumerationValue bTypeValue =
    database.EnumerationSets["Building Type"]["Residential"];
  meter001.Attributes["Building Type"].SetValue(new AFValue(bTypeValue));
  meter001.Attributes["City"].SetValue(new AFValue("London"));
}
```

7. Create the root element called `Geographical Locations`. Then, create three child elements called `London`, `Montreal`, and `San Francisco`, based on the District template.

```
private static void CreateCityElements(AFDatabase database)
{
  if (database == null) return;

  if (!database.Elements.Contains("Geographical Locations"))
  {
    AFElement geoLocations = database.Elements.Add("Geographical Locations");
    AFElementTemplate cityTemplate = database.ElementTemplates["City"];
    geoLocations.Elements.Add("London", cityTemplate);
    geoLocations.Elements.Add("Montreal", cityTemplate);
    geoLocations.Elements.Add("San Francisco", cityTemplate);
  }

  if (database.IsDirty)
```

```
    database.CheckIn();
}
```

8. Add meter elements as weak references under the District elements, following the reference database.

```
private static void CreateWeakReferences(AFDatabase database)
{
  if (database == null) return;

  AFReferenceType weakRefType = database.ReferenceTypes["Weak Reference"];
  AFElement meters = database.Elements["Meters"];
  AFElement locations = database.Elements["Geographical Locations"];
  AFElementTemplate cityTemplate = database.ElementTemplates["City"];

  foreach (AFElement meter in meters.Elements)
  {
    string cityName = meter.Attributes["City"].GetValue().ToString();
    if (string.IsNullOrEmpty(cityName))
      continue;
    AFElement city = locations.Elements[cityName];
   if (city == null)
      locations.Elements.Add(cityName, cityTemplate);
   if (!city.Elements.Contains(meter.Name))
     city.Elements.Add(meter, weakRefType);
   }

   if (database.IsDirty)
     database.CheckIn();
}
```

Lesson 5: Working with AF event frames

After completing this lesson you will be able to:

- Use AF SDK to manage event frames
- Create an event frame template
- Create and work with event frames

Help for methods and data types used in this lesson

- AFEventFrameSearch.FindEventFrames() Method (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSIsoft_AF_Search_AFEventFrameSearch_FindEventFrames.htm)
- AFEventFrame.CaptureValues() Method (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSIsoft_AF_EventFrame_AFEventFrame_CaptureValues.htm)
- AFEveventFrame Class (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/T_OSIsoft_AF_EventFrame_AFEventFrame.htm)

Introduction

An Event Frame represents a time period defined by a start time and end time. Each Event Frame typically references one or more PI AF elements to associate the event with an asset. An Event Frame can also have a collection of child Event Frames, which represent child events that make up the larger event.

When creating `AFEventFrame` objects, the following properties should usually be set: **`StartTime`**, **`EndTime`**, and **`PrimaryReferencedElement`**.

```
AFEventFrame ef = new AFEventFrame(database, "*", eventFrameTemplate);
ef.SetStartTime(startTime);
ef.SetEndTime(endTime);
ef.PrimaryReferencedElement = meter;
```

When creating Event Frame templates in code, you might be surprised to discover there is no such object as `AFEventFrameTemplate`. Instead, an Event Frame template is a simply an `AFElementTemplate` with its `InstanceType` property set to `typeof(AFEventFrame)`.

```
AFElementTemplate eventFrameTemplate =
  database.ElementTemplates["Name of Template"];
if (eventFrameTemplate == null)
AFElementTemplate eventFrameTemplate =
  database.ElementTemplates.Add("Name of Template");
eventFrameTemplate.InstanceType = typeof(AFEventFrame);
```

To find existing PI AF Event Frames, use the `AFEventFrameSearch.FindEventFrames()` method. You can search by name, start time, end time, duration, template, category, and a variety of other fields. Note that unlike elements, event frames cannot be accessed directly as a property under the `AFDatabase` object. For more information, see the online AF SDK reference AFEventFrameSearch.FindEventFrames() (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/Overload_OSIsoft_AF_EventFrame_AFEventFrame_FindEventFrames.htm).

Note that `AFEventFrameSearch` is analogous to `AFElementSearch` from Lesson 2.

```
AFEventFrameSearch eventFrameSearch =
  new AFEventFrameSearch(database, "EventFrame Captures",
AFEventFrameSearchMode.ForwardFromStartTime,
  startTime, queryString);
```

Because Event Frames typically represent historical time ranges, attribute values of event frames typically do not change over time and can be cached to improve performance. In AF SDK, you can call the `CaptureValues()` method on an event frame instance to capture and save attribute values to the PI AF server for faster subsequent retrieval. Updating the start or end time of the event frame will automatically recapture the values. For more information, see the online AF SDK reference CaptureValues() (https://techsupport.osisoft.com/Documentation/PI-AF-SDK/html/M_OSIsoft_AF_EventFrame_AFEventFrame_CaptureValues.htm).

```
foreach (AFEventFrame item in eventFrameSearch.FindEventFrams())
{
    item.CaptureValues();
}
```

Lesson 5 exercises

Your next task is to develop an energy reporting system for the PI AF database Green Power Company. You should familiarize yourself with the PI AF database using PI System Explorer before starting the exercise.

The Green Power Company database contains 12 electric meters deployed to different buildings. Your task is to create PI AF Event Frames to track average energy usage per day. Each PI AF Event Frame is created using a PI AF Event Frame Template. The template should have one attribute configured to report the average of the Energy Usage attribute of a meter over the event frame time range.

Please implement the following methods inside `Program5.cs` and run the application with the arguments provided for each of the following methods in `Main()`.

For each of the following exercises, your task is to add the logic to each method. Add the methods to your console application and test each one by writing output to the console.

- **Exercise 1**

  Create an AF Event Frame template called `Daily Usage`. Event frame names created from this template should include the event frame template name, referenced element name, and start date. For example: `Usage Tracker-Meter002-2016-02-01 00:00:00`.

  Create a PI AF Attribute Template under the event frame template called `Average Energy Usage`. Configure the `AFAttributeTemplate` properties such that the event frame attribute reports the average value of the referenced Energy Usage attribute over the event frame time range. See Lesson 5 hints and tips for an example configuration string to use.

  ```
  AFElementTemplate CreateEventFrameTemplate(AFDatabase database)
  ```

- **Exercise 2**

  For each meter and for each day within February 1 - 5, 2016, create an AFEventFrame based off of the "Daily Usage" template. Set the PrimaryReferencedElement to the appropriate meter.

```
void CreateEventFrames(AFDatabase database, AFElementTemplate
eventFrameTemplate)
```

- **Exercise 3**

  Save the value of Average Energy Usage attribute to the PI AF server for each event frame using `CaptureValues()` on the event frame instance. You will first need to find all event frames using the `AFEventFrameSearch.FindEventFrames()` method.

  ```
  void CaptureValues(AFDatabase database, AFElementTemplate eventFrameTemplate)
  ```

- **Exercise 4**

  Find all of the event frames referencing Meter003. Use the `AFEventFrameSearch.FindEventFrames()` method. Then print to the console the event frame name, primary referenced element name, and value of the Average Energy Usage attribute.

  ```
  void PrintReport(AFDatabase database, AFElementTemplate eventFrameTemplate)
  ```

  When the method is run, it should product the following output:

  ```
  Daily Usage-Meter003-2017-06-29-EF, Meter003, 306.564053292511
  Daily Usage-Meter003-2017-06-29-EF1, Meter003, 306.564053292511
  Daily Usage-Meter003-2017-06-30-EF, Meter003, 296.339212218409
  Daily Usage-Meter003-2017-06-30-EF1, Meter003, 296.339212218409
  Daily Usage-Meter003-2017-07-01-EF, Meter003, 297.125675030912
  Daily Usage-Meter003-2017-07-01-EF1, Meter003, 297.125675030912
  Daily Usage-Meter003-2017-07-02-EF, Meter003, 303.961577200124
  Daily Usage-Meter003-2017-07-02-EF1, Meter003, 303.961577200124
  Daily Usage-Meter003-2017-07-03-EF, Meter003, 296.181190443142
  Daily Usage-Meter003-2017-07-03-EF1, Meter003, 296.181190443142
  Daily Usage-Meter003-2017-07-04-EF, Meter003, 296.944001749179
  Daily Usage-Meter003-2017-07-04-EF1, Meter003, 296.944001749179
  Daily Usage-Meter003-2017-07-05-EF, Meter003, 294.295850664074
  Daily Usage-Meter003-2017-07-05-EF1, Meter003, 294.295850664074
  ```

Lesson 5 hints and tips

`CaptureValues()` is used only on closed event frames; that is, an Event Frame with an **`EndTime`** that occurs before `AFTime.MaxValue`. Event frames that are not closed have values that can still change. If the `CaptureValues()` method is called and the **`EndTime`** has not been set, then an exception is thrown.

Use the template `NamingPattern` property to define the event frame names. To obtain an understanding of how the template works, try setting it in PI System Explorer first before using it in AF SDK. For example, try the template shown here:

```
TEMPLATE%-%ELEMENT%-%STARTTIME:yyyy-MM-dd%-EF*
```

To configure the event frame attribute to report an average, set the data reference to a PI Point DR. Use the following ConfigString:

```
.\Elements[.]|Energy Usage;TimeRangeMethod=Average
```

The ConfigString shown above configures the Average Energy Usage attribute to report the average of the primary referenced element's Energy Usage over the event frame's time range.

Lesson 5 exercise solutions

- **Exercise 1: Create Event Frame Template**

```
static AFElementTemplate CreateEventFrameTemplate(AFDatabase database)
{
  AFElementTemplate eventFrameTemplate =
    database.ElementTemplates["Daily Usage"];
  if (eventFrameTemplate != null)
    return eventFrameTemplate;

  eventFrameTemplate = database.ElementTemplates.Add("Daily Usage");
  eventFrameTemplate.InstanceType = typeof(AFEventFrame);
  eventFrameTemplate.NamingPattern =
    @"%TEMPLATE%-%ELEMENT%-%STARTTIME:yyyy-MM-dd%-EF*";

  AFAttributeTemplate usage =
    eventFrameTemplate.AttributeTemplates.Add("Average Energy Usage");
  usage.Type = typeof(Single);
  usage.DataReferencePlugIn = AFDataReference.GetPIPointDataReference();
  usage.ConfigString = @".\Elements[.]|Energy Usage;TimeRangeMethod=Average";
  usage.DefaultUOM = database.PISystem.UOMDatabase.UOMs["kilowatt hour"];

  if (database.IsDirty)
    database.CheckIn();

  return eventFrameTemplate;
}
```

- **Exercise 2: Create Event Frames**

```
static void CreateEventFrames(AFDatabase database, AFElementTemplate
eventFrameTemplate)
{
  string queryString = "Template:MeterBasic";
  {
    // This method returns the collection of AFBaseElement objects that
    // were created with this template.
    using (AFElementSearch elementQuery =
      new AFElementSearch(database, "Meters", queryString))
    {
       DateTime timeReference = DateTime.Today.AddDays(-7);
       int count = 0;
       foreach (AFElement meter in elementQuery.FindElements())
       {
         foreach (int day in Enumerable.Range(1, 7))
         {
           AFTime startTime = new AFTime(timeReference.AddDays(day - 1));
           AFTime endTime = new AFTime(startTime.LocalTime.AddDays(1));
           AFEventFrame ef = new AFEventFrame(database, "*",
             eventFrameTemplate);
           ef.SetStartTime(startTime);
           ef.SetEndTime(endTime);
           ef.PrimaryReferencedElement = meter;
           // It is good practice to periodically
           // check in the database
           if (++count % 500 == 0)
              database.CheckIn();
         }
       }
    }
   }

   if (database.IsDirty)
```

```
    database.CheckIn();
}
```

- **Exercise 3: Capture Values**

```
static public void CaptureValues(AFDatabase database, AFElementTemplate
eventFrameTemplate)
{
  // Formulate search constraints on time and template
  AFTime startTime = DateTime.Today.AddDays(-7);
  string queryString = $"template:\"{eventFrameTemplate.Name}\"";
  using (AFEventFrameSearch eventFrameSearch =
    new AFEventFrameSearch(database, "EventFrame Captures",
    AFEventFrameSearchMode.ForwardFromStartTime, startTime, queryString))
    {
      eventFrameSearch.CacheTimeout = TimeSpan.FromMinutes(5);
      int count = 0;
      foreach (AFEventFrame item in eventFrameSearch.FindEventFrames())
      {
         item.CaptureValues();
         if ((count++ % 500) == 0)
            database.CheckIn();
      }

      if (database.IsDirty)
         database.CheckIn();
    }
}
```

- **Exercise 4: Print Report**

```
static void PrintReport(AFDatabase database, AFElementTemplate
eventFrameTemplate)
{
   AFTime startTime = DateTime.Today.AddDays(-7);
   AFTime endTime =
     startTime.LocalTime.AddDays(+8); // Or DateTime.Today.AddDays(1);
   string queryString =
     $"template:'{eventFrameTemplate.Name}' ElementName:Meter003";
   using (AFEventFrameSearch eventFrameSearch =
     new AFEventFrameSearch(database, "EventFrame Captures",
     AFSearchMode.StartInclusive, startTime, endTime, queryString))
   {
     eventFrameSearch.CacheTimeout = TimeSpan.FromMinutes(5);
     foreach (AFEventFrame ef in eventFrameSearch.FindEventFrames())
     {
        Console.WriteLine("{0}, {1}, {2}",
        ef.Name,
        ef.PrimaryReferencedElement.Name,
        ef.Attributes["Average Energy Usage"].GetValue().Value);
     }
   }
}
```

Conclusion

Congratulations! By now you should have a good, basic understanding of how to use AF SDK. Before you go, please be sure to visit the PI Developers Club space within PI Square: PI Square (https://pisquare.osisoft.com). You can find much more information about the PI system, software development, and practical knowledge from experts.

If you are viewing this guide online, be sure to notice and use the link at the bottom of each page, which asks: Was this information helpful? OSIsoft values your input and will make every effort to improve and enhance this information based on your feedback.

Technical support and other resources

For technical assistance, contact OSIsoft Technical Support at +1 510-297-5828 or through the OSIsoft Tech Support Contact Us page (https://techsupport.osisoft.com/Contact-Us/). The website offers additional contact options for customers outside of the United States.

When you contact OSIsoft Technical Support, be prepared to provide this information:

- Product name, version, and build numbers
- Details about your computer platform (CPU type, operating system, and version number)
- Time that the difficulty started
- Log files at that time
- Details of any environment changes prior to the start of the issue
- Summary of the issue, including any relevant log files during the time the issue occurred

To ask questions of others who use OSIsoft software, join the OSIsoft user community, PI Square (https://pisquare.osisoft.com). Members of the community can request advice and share ideas about the PI System. The PI Developers Club space within PI Square offers resources to help you with the programming and integration of OSIsoft products.