2016 OSIsoft TechCon

PI AF SDK for Beginners

OSIsoft, LLC
777 Davis St., Suite 250
San Leandro, CA 94577 USA
Tel: (01) 510-297-5800
Web: http://www.osisoft.com

Published: March 21, 2016

Table of Contents

Introduction

The PI AF SDK is a .NET-based programmatic library that provides access to the variety of data stored in the PI System including:

- Assets and their properties and relationships
- Time series data from the PI Data Archive and other sources
- Event frames that record and contextualize process events

This course will cover basic usage of the PI AF SDK. A background in .NET application development and a familiarity with the PI System (but not PI AF SDK) are assumed. An additional short introduction with reference links for PI AF SDK is available in Appendix 1 if further background is needed.

Getting Started

1. Open the Visual Studio Project

Open the file:

Desktop>PI AF SDK for Beginners>Exercises-PI-AF-SDK-For-Beginners>
`PI-AF-SDK-For-Beginners-TechCon2016.sln`

Each exercise is contained in a Console project. The project consists of `ProgramN.cs` (where N = 1 to 5) which contains a series of calls to stub methods for each part of the exercise. At the end of `Main()`, it prompts for any key so that output remains in the console window until exit or the debug run is halted. An accompanying "solution" project (project name appended with "-Sln") exists for each exercise.

Each exercise consists of the following sections:

1. **Introduction** – Discuss the features and methods used in the exercise.
2. **Problem** – Describes a series of objectives to achieve. Not all need be completed.
3. **Hints** (optional) – Gives some hints to help complete the problem.
4. **Example** (optional) – Shows some example code that may be useful.
5. **Discussion** (optional) – Follow-up on additional concepts. Optional to read.

Look at the answers if you get stuck. This is not an exam. ☺

Run specific projects by right-clicking the desired project, scroll to "Debug", expand to "Start new instance". Otherwise, use "Set as Startup project" and F5 to start.

2. Understand the AF Database structure

All exercises are based on the AF Database called "**Magical Power Company**". This database contains a list of meters that have been deployed to various building sites. The AF hierarchy is organized based on the geographic region of the buildings.

Please familiarize yourself with this AF Database using PI System Explorer before starting the exercises.

3. Exercise difficulty

A list with approximate difficulty level (1 = easier) is below:

Exercise	**Difficulty**
Ex1-Connection-And-Hierarchy-Basics	1
Ex2-Searching-For-Assets	2
Ex3-Reading-And-Writing-Data	2
Ex4-Building-An-AF-Hierarchy	3
Ex5-Working-With-Event-Frames	3

Exercise 1: Connection and Hierarchy Basics

Introduction

The first step in a PI AF SDK application is connecting to an AF Server or PI Data Archive.

The list of AF Servers known by the client is represented as `PISystems`. The list of PI Data Archive servers known by the client is represented as `PIServers`. Below, we show how to obtain an object representing the AF Server called “My AF Server” and PI Data Archive called “My PI Data Archive”.

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

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

Once we have a reference to the `PISystem`, we can obtain a reference to its collection of `AFDatabase` objects, via `piSystem.Databases`. To get a specific `AFDatabase`, we can use the `AFDatabases` indexer and pass in a name.

```
AFDatabase db = piSystem.Databases["Meters"];
```

From the `AFDatabase`, we can further obtain references to other collections such as root elements, categories, templates, and tables. To get a specific object from a collection, we can use the collection’s indexer and pass in a name.

```
AFElements rootElements = db.Elements;
AFElement element1 = rootElements["Meter1"];
AFCategories elementCategories = db.ElementCategories;
AFElementTemplates elementTemplates = db.ElementTemplates;
AFTables tables = db.Tables;
```

The object hierarchy maps very closely to what is seen in PI System Explorer, so it is an excellent visual guide for illustrating object relationships within PI AF SDK.

Note that we do not have to explicitly call a `Connect()` method. PI AF SDK will perform the connection automatically as required. This is called implicit connection. If we want to make an explicit connection, we can call `Connect()` on a `PISystem` instance. Implicit and explicit connections are discussed in detail in the PI AF SDK Online Reference.

Problem

You have been accepted into PI School of Witchcraft and Wizardry, and you are taking their Introduction to PI Wizard Programming class. Although non-wizards have known about the value of PI for decades, only recently have wizards discovered the value of a PI System infrastructure.

Your assignment will use the AF Database “**Magical Power Company**” on your local AF Server. You will be asked to print the names and properties of various AF objects to the Console. Please familiarize yourself with this AF Database using PI System Explorer before starting the exercises.

Please implement the following methods inside `Program1.cs` and run the application with the arguments provided for each of the following methods in `Main()`.

Instructor-Led

1) `void PrintRootElements(AFDatabase database)`

   Print the name of each `AFElement` directly under the AF Database root.

Student Exercises

2) `void PrintElementTemplates(AFDatabase database)`

   Print the name of each `AFElementTemplate`. For each element template, also print out the name of each `AFCategory` in its `Categories` collection.

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

   Print the name of each `AFAttributeTemplate` for the passed in element template. For each attribute template, also print out the name of its data reference plugin.

4) `void PrintEnergyUOMs(PISystem system)`

   Print the name of each UOM and its abbreviation under the `UOMClass` "Energy".

5) `void PrintEnumerationSets(AFDatabase database)`

   Print the name of each `AFEnumerationSet`. For each set, print out its list of states.

6) `void PrintCategories(AFDatabase database)`

   Print the name of each element category and attribute category.

Discussion

Client-side caching and reconnection

PI AF SDK internally maintains a cache for each user. This cache is initialized upon first connection. Under the hood, PI AF SDK will allow connections to the AF Server and PI Data Archive to drop after periods of inactivity and will automatically reestablish them when activity is resumed. The user cache is preserved between periods of inactivity and can be reused after reconnection. Calling `Disconnect()` explicitly, however, will clear out the user cache.

Therefore, we do not recommend users to manually incorporate their own reconnection logic using `Disconnect()/Connect()` on a `PISystem` or `PIServer` instance, since this will force an expensive cache re-initialization upon reconnection. In most of your projects with the PI AF SDK, using implicit connections and having PI AF SDK handle reconnections should suffice.

Connecting to collectives

Connecting to AF Server and PI Data Archive collectives work similarly in PI AF SDK, so we will focus only on AF Server in this discussion. It is important to keep in mind that each `PISystem` object represents a *logical* AF Server, which may or may not be a collective. To check if the PI System is a collective, check its `Collective` property, which returns an `AFCollective` object.

Each `AFCollective` object exposes a list of collective members, via the `Members` property. The `Connect()` method can be called on an `AFCollectiveMember` to explicitly connect to a specific physical AF Server.

When making an explicit connection to `PISystem` instance that is a collective, the `AFConnectionPreference` enumeration determines which physical AF Server should the connection be tried against first. The connection preference is checked in the following places in order of priority

1) `Connect()` overload on the `PISystem` instance
2) `ConnectionInfo.Preference` setting on the `PISystem` instance
3) `AFConnectionInfo.DefaultPreference` setting (default: `PreferPrimary`)

The `Priority` property of the `AFCollectiveMember` is used to determine which server should be attempted for connection when the preference is set to `Any` or `PreferPrimary`. The highest priority is given a priority of 1. Note a secondary server can be given a priority of 1. A negative priority indicates connections to that member are not allowed. By setting priorities appropriately on user machines and running HA aware applications that use `Any` as a connection preference, the load on a collective can be distributed among its members.

```
// Explicitly connect to server named AFServer3
AFCollectiveMember member3 = piSystem.Collective.Members["AFServer3"];
member3.Connect();

// Use member priority to select server
piSystem.Connect(null, AFConnectionPreference.Any);
```

Exercise 2: Searching for Assets

Introduction

In this exercise, we learn how to search for assets (elements and their attributes) using PI AF SDK. We learn about different search criteria that can be used. We learn how to handle search results and how to page through search results when the result set is large.

If there is one lesson to take away from this exercise, it would be:

Reduce the search results to a minimum on the server, not on the client.

This means calling PI AF SDK methods that allow the server to process the search instead of writing custom search logic on the client.

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

Problem

You have graduated from PI School of Witchcraft and Wizardry and have been hired as a Developer by Magical Power Company. Your assignment is to develop a client application that allows users to search for meters (AF Elements) and attributes. In the application, the user can search for meters by name, element template, element category, and their attribute values. The user can also search for meter attributes by name, element template, and attribute category.

Your assignment will use the AF Database "**Magical Power Company**" on your local AF Server. You will be asked to write several methods to search for elements and attributes and print the search results to the Console. Please familiarize yourself with this AF Database using PI System Explorer before starting the exercise.

Please implement the following methods inside `Program2.cs` and run the application with the arguments provided for each of the following methods in `Main()`.

Instructor-Led

1) `void FindMetersByName(AFDatabase database, string elementNameFilter)`

   Print the following information for each found meter: Name, Element Template, Element Categories

Student Exercises

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

Find all meters 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 found.

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

Find all meters supplied by the provided substation. Then print the names of each found meter.

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

Find all meters with Energy Usage above the supplied limit. Then print the names of each found meter.

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

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

See Hints and Examples below for guidance.

Hints

Use the "Search Query Syntax Overview" help page in %pihome%\help\AFSDK.chm to help form a string for `AFElementSearch()` (new in AF 2016).

For 5), try AFAttribute.FindElementAttributes.

Examples

Using search for AFElements

```
AFElementSearch elementquery = new AFElementSearch(database, "TemplateSearch",
string.Format("template:\"{0}\"", templateName));
foreach (AFElement element in elementquery.FindElements())
{ /* do work */ }
```

Using AFAttribute.FindElementAttributes

```
AFCategory attrCat = _database.AttributeCategories["Location"];

AFNamedCollectionList<AFAttribute> foundAttributes = AFAttribute.FindElementAttributes(
       database: _database,
       searchRoot: null,
       nameFilter: "*",
       elemCategory: null,
       elemTemplate: elemTemp,
       elemType: AFElementType.Any,
       attrNameFilter: "*",
       attrCategory: attrCat,
       attrType: TypeCode.Empty,
       searchFullHierarchy: true,
       sortField: AFSortField.Name,
       sortOrder: AFSortOrder.Ascending,
       maxCount: 100);
```

Discussion

Paged Searching

Network bandwidth and latency between client and AF Server will affect search performance. AF search methods provide facilities to retrieve paged results. For the new `AFSearch` methods in AF 2016 and later, consult the documentation for `AFGlobalSettings.CollectionPageSize`, `FindElements()` argument 'pageSize' for defaults and methods to modify page size. Additionally, for long queries in which `AFElements` may be modified as the search results are returned, examine how `AFSearch.CacheInterval` affects frequency at which search results are refreshed.

The example 5 above (and exercise solution) limited the maximum number of results to 100 (`maxCount` argument). For large AF databases, we may then miss results. Further, even if we did return all the results now, there is no guarantee that the database (and number of results) won't grow in the future.

One tempting workaround is to simply increase `maxCount` to a very large number. However, how "large" should it be? Even if we knew of an upper limit, this puts us at risk of our query timing out.

To overcome these issues, it is recommended to perform *paged* searches instead. We only get results limited by the page size, but we can then query for the next (or previous) page as desired. In a paged call, we specify a `startIndex` and `pageSize`. We also provide a reference to the `totalCount` that will be set by the paged search, so we know when to stop paging. In the Online PI AF SDK Reference, try to find the overloads of the methods above that support paged queries.

A general pattern for getting paged results is shown below:

```
const int pageSize = 100;
int startIndex = 0;
int totalCount;
AFNamedCollectionList<AFElement> myElements;
do
{
        myElements = AFElement.FindElements(
                            ... argument list omitted for brevity
                            startIndex: startIndex,
                            pageSize: pageSize,
                            totalCount: out totalCount);
        if (myElements == null) break;
        ProcessElements(myElements);

        startIndex += pageSize; // Advance to next page.
} while (startIndex < totalCount);
```

As a bonus, try to improve our existing solution for this exercise to use paged searching instead. As an extra bonus, convert the paging to an iterator pattern using the new `AFElementSearch` in AF 2016, which is more performant.

The previously available search methods (pre AF 2016) in the table below are still available. Examples for these methods may be found in the Online PI AF SDK Reference's Search Example. The table includes some usage scenarios for each method.

Method	Returns	Name	Description	Search Root	Template	Category	Reference Type	Element Type	Paths	Attribute Value	Relative Object	Max Bandwidth	Caveats
FindElementsByPath	Element								X		X	1000 /sec	Portions execute on the client
FindElements	Element	X		X	X	X		X				20k /sec	
FindElements	Element	X	X	X	X	X						20k /sec	
FindElementsByAttribute	Element	X		X						X		20k /sec	Index attributes for best performance
FindElementsByTemplate	Element			X	X							20k /sec	
FindElementsByCategory	Element			X		X						20k /sec	
FindElemenstByReferenceType	Element			X			X					20k /sec	
FindElementAttributes	Attribute	X		X	X	X		X				20k /sec	
FindAttributesByPath	Attribute								X		X	1000 /sec	Portions execute on the client

From PI AF SDK – Performance and Scalability, vCampus Live 2012.
Note: *ByPath performance is improved with AF 2.6 (2014).

You are encouraged to refer to the Online PI AF SDK Reference to learn more about these methods.

Exercise 3: Reading and Writing Data

Introduction

In this exercise, we learn how to read data from and write data to PI AF. All events (timestamp, value) in PI AF SDK are represented as an `AFValue`. The `AFValue` contains two important properties of interest:

`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

The `AFTime` is similar to a .NET `DateTime` object. One main difference is that `AFTime` objects can be constructed via PI Time syntax (e.g. 't', '*-10m').

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

```
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);
```

Bulk Calls

If we want to retrieve data within the same time range for many attributes, it is time-consuming to make a data call for each attribute, because we will make many roundtrips to the PI Data Archive. In networked systems, the latency cost is typically the largest.

We can reduce the number of roundtrips using list calls in PI AF SDK. First, we can create a list of attributes for which we want data on the client, storing them in an `AFAttributeList`. We then access the `Data` property on the `AFAttributeList`, which exposes data retrieval methods for the entire attribute list. Now, we can make one call to the server to retrieve data for all the attributes.

```
AFAttributeList attrList = AFAttribute.FindElementAttributes(
       // argument list omitted for brevity
       );
// Bulk call to get value at provided time for all found attributes above
IList<AFValue> vals = attrList.Data.RecordedValue(time);
```

Problem

You have recently been promoted to Senior Developer at Magical Power Company. Your assignment is to develop a client application that allows users to read data from and write data to the PI Data Archive.

Your assignment will use the AF Database "**Magical Power Company**" on your local 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 as well. Please familiarize yourself with this AF Database using PI System Explorer before starting the exercise.

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

Instructor-Led

1) `void PrintHistorical(AFDatabase database, string meterName, string startTime, string endTime)`

   This method should retrieve historical 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 in UTC, Value in kilojoule

Student Exercises

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

   This method should retrieval 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 hour.

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

   This method should 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 hour.

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

   This method should 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.

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

   This method should retrieve 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 hour.

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

   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 above 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 2 `UpdateValues` calls. What are some tradeoffs when limiting the number of remote calls in this case? How would you ensure no data loss?

See Hints and Examples below for guidance.

Hints

For 4) and 5), try AFAttribute.FindElementAttributes to obtain an `AFAttributeList`. Then, use `AFAttributeList.Data` to make the bulk calls.

For 5), do not use `RecordedValues()` and compute the average in your code. Instead, use a `Summaries()` method to allow the PI Data Archive to compute the average and return only the average to the client.

For 6), the existing archived values must be removed before inserting new values (for PI AF SDK 2.7.x and below). Starting from 2.8, you may use `ReplaceValues()`.

Examples

Get hourly average from the PI Data Archive

```
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}, Value (kilowatt hour): {1}",
	val.Timestamp.LocalTime, val.Value);
}
```

Write values to multiple attributes

```
// Generate some values for attribute1
List<AFValue> vals1 = Enumerable.Range(1, 10)
	.Select(x => new AFValue(attribute1, x, new AFTime(DateTime.Today.AddSeconds(x))))
	.ToList();

// Generate some values for attribute2
List<AFValue> vals2 = Enumerable.Range(1, 10)
	.Select(x => new AFValue(attribute2, x, new AFTime(DateTime.Today.AddSeconds(x))))
	.ToList();
```

```
List<AFValue> valsCombined = vals1;
valsCombined.AddRange(vals2);

// Write to both attributes
AFListData.UpdateValues(valsCombined, AFUpdateOption.Insert);
```

Remove values

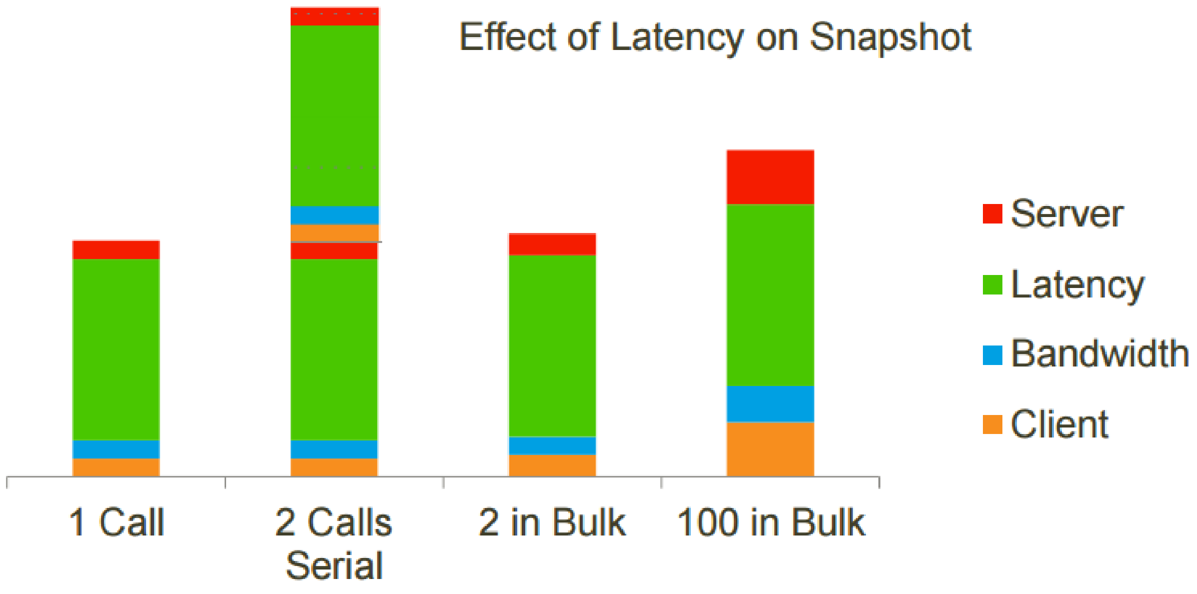
```
AFValues valsToRemove = attribute.Data.RecordedValues(
        timeRange: timeRange,
        boundaryType: AFBoundaryType.Inside,
        desiredUOM: null,
        filterExpression: null,
        includeFilteredValues: false);

AFListData.UpdateValues(valsToRemove, AFUpdateOption.Remove);
```

Discussion

Network Latency

It is almost always favorable to use bulk calls to read and write data. This is because network latency is almost always the largest bottleneck. Use `AFAttributeList.Data` and its methods to read data. Use `AFListData.UpdateValues()` to write data. Serialized calls for each PI Point (e.g. `AFAttribute.Data` methods) incur the latency cost for every point; bulk calls spread this cost over many PI Points.

Bulk vs. Serial – Performance 101



From Optimizing PI AF Server and PI AF SDK Applications, vCampus Live 2013.

Summary calls

New users to the PI System are often unaware of the power of the PI Data Archive to perform statistical calculations, such as finding the minimum, maximum, average, and total value of a PI Point over a period of time. As a result, all of the raw data is queried and returned from the PI Data Archive, and the client code implements logic to compute statistical quantities. This can be harmful in two major ways:

1) A large amount of data can be transferred over the network, saturating bandwidth.
2) Client machines typically have fewer CPU cores to perform calculations. As a result, application performance and user experience will suffer.

The `Summary()` and `Summaries()` methods on `AFAttribute.Data` and `AFAttributeList.Data` solve these issues by allowing the PI Data Archive to perform the calculations and return only the requested averages. Therefore, when querying for statistical quantities, always check if these methods suit your needs.

In some cases, client-side summaries are the only option. Instead of implementing your own logic, check if the `AFValues.Summary()` or `AFValues.Summaries()` suit your needs. These methods mimic the server-side algorithm and can save you time from re-inventing the wheel or having inconsistent code bases.

Exercise 4: Building an AF Hierarchy

Introduction

In this exercise, we will build an AF hierarchy from scratch. We will create an AF database, create element and attribute templates, create an asset hierarchy, and link to PI Points from the attribute configuration. We will also learn about recommended PI AF SDK patterns such as checking in changes in bulk.

Below is a simple example for creating a new AF Database, AF Element, and AF Attribute, and checking in the changes. Refer to the Examples section below for more detailed examples.

```
PISystem piSystem = new PISystems().DefaultPISystem;
AFDatabase newDb = piSystem.Databases.Add("MyDb");
AFElement newEl = newDb.Elements.Add("MyElement");
AFAttribute newAttr = newEl.Attributes.Add("MyAttribute");
newDb.CheckIn();
```

Problem

You have recently been promoted to Staff Developer at Magical Power Company. The company would like to also include Feeders in their AF Database. Your assignment is to create templates, elements, and attributes for feeder assets.

Please implement the following methods inside `Program4.cs` and run the application with the arguments provided for each of the following methods in `Main()`.

Instructor-Led

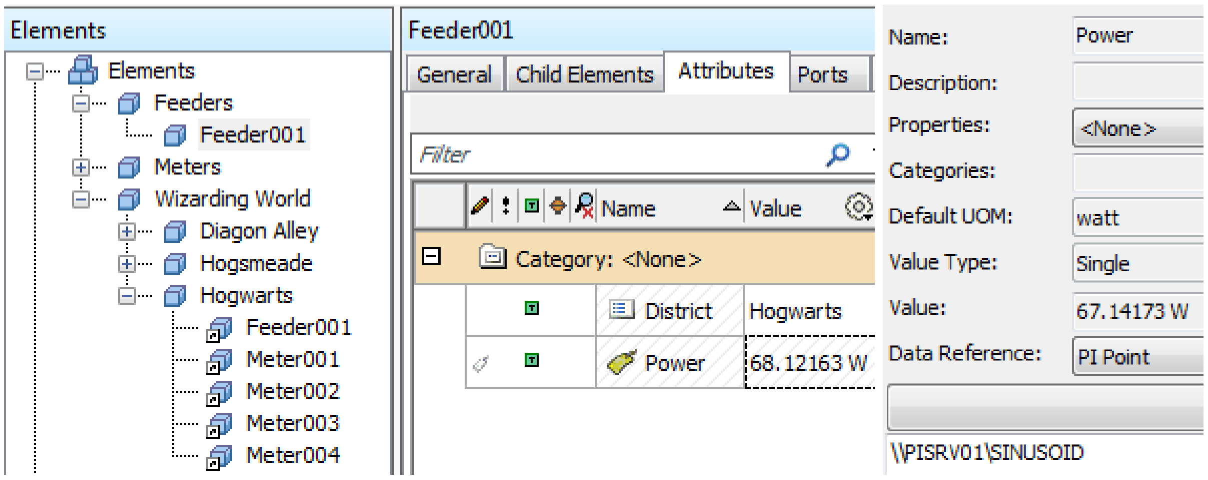
1) Create an element template called *FeederTemplate*. Create one attribute template called *District* with no data reference. Create another attribute template called *Power*. Set the DefaultUOM to “watt” and the Type to Single. Set the data reference to “PI Point”.

Student Exercises

2) Create an element called *Feeders* directly under the database root.

3) Create an element called *Feeder001* under *Feeders* based off of the *FeederTemplate* element template. Set the value of the *District* attribute to “Hogwarts”. Set the ConfigString property of the *Power* attribute such that the attribute’s data source is the “SINUSOID” PI Point.

4) Create a weak reference between *Feeder001* and the *Hogwarts* element. The final result should look like this:



Elements

- Elements
  - Feeders
    - Feeder001
  - Meters
  - Wizarding World
    - Diagon Alley
    - Hogsmeade
    - Hogwarts
      - Feeder001
      - Meter001
      - Meter002
      - Meter003
      - Meter004

Feeder001

Name	Value
Category: <None>	
District	Hogwarts
Power	68.12163 W

Name:	Power
Description:	
Properties:	<None>
Categories:	
Default UOM:	watt
Value Type:	Single
Value:	67.14173 W
Data Reference:	PI Point

\\PISRV01\SINUSOID

Bonus

Create a replica of the AF Database “Magical Power Company”.

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

   Create an AF Database called “**Mythical Power Company**”.

2) `void CreateCategories(AFDatabase database)`

   Create the element categories and attribute categories.

3) `void CreateEnumerationSets(AFDatabase database)`

   Create the enumeration sets and their values.

4) `void CreateTemplates(AFDatabase database)`

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

5) `void CreateElements(AFDatabase database)`

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

6) `void SetAttributeValues(AFDatabase database)`

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

7) `void CreateDistrictElements(AFDatabase database)`

   Create the root element called “Wizarding World”. Then, create three child elements called “Diagon Alley”, “Hogsmeade”, and “Hogwarts”, based on the District template.

8) `void CreateWeakReferences(AFDatabase database)`

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

See Hints and Examples below for guidance.

Examples

Create an AFDatabase

```
PISystem piSystem = new PISystems()["AFServerName"];
AFDatabase database = piSystem.Databases.Add("Hogwarts");
```

Create an AFElementTemplate

```
AFElementTemplate elemTemp = database.ElementTemplates.Add("House");
```

Create an AFAttributeTemplate

```
AFAttributeTemplate attrTemp = elemTemp.AttributeTemplates.Add("Average Temperature");
```

Create an AFEnumerationSet

```
AFEnumerationSet enumSet = database.EnumerationSets.Add("Modes");
enumSet.Add("Manual", 0);
enumSet.Add("Auto", 1);
```

Set AFAttributeTemplate properties

```
// Set the data reference
AFPlugIn piPointDR = AFDataReference.GetPIPointDataReference(piSystem);
attrTemp.DataReferencePlugIn = piPointDR;
// Set the Type
attrTemp.Type = typeof(double);
// Set the ConfigString
attrTemp.ConfigString = @"\\server\%Database%.%Element%.%Attribute%";
// Set the UOM
UOM celsius = _database.PISystem.UOMDatabase.UOMs["degree Celsius"];
houseAvgTemp.DefaultUOM = celsius;
// Set to enumeration set
attrTemp2.TypeQualifier = enumSet;
```

Create Elements

```
// From template
AFElement newElement = database.Elements["ParentElem"].Elements.Add("ChildElem",
elemTemp);
// Without template
AFElement newElement = database.Elements["ParentElem"].Elements.Add("ChildElem");
```

Create an AFEnumerationSet

```
AFEnumerationSet enumSet = database.EnumerationSets.Add("Modes");
enumSet.Add("Manual", 0);
enumSet.Add("Auto", 1);
```

Set AFAttributeTemplate type to be an enumeration set

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

Check in changes

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

Discussion

Commit changes using bulk CheckIn

In the exercise solution code, we 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 since each will cause a call to the SQL server. When making multiple modifications, a general rule of thumb is to `CheckIn` 200 objects at a time. If more control is required over the list of objects to be checked in, you can use the `CheckIn()` method on a `PISystem` instance (PISystem.CheckIn()).

Add Overloads

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

1) 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");
```

2) Methods that accept a reference to an existing `AFElement` object and return void.

A common use case for 2) is when adding an existing element as a weak reference to a parent element.

```
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);
```

Exercise 5: Working with AF Event Frames

Introduction

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

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

When creating AF Event Frame templates, use the same approach as creating AF Element Templates, but set the `InstanceType` property of the `AFElementTemplate` object to `typeof(AFEventFrame)`.

To find existing AF Event Frames, use AFEventFrameSearch.FindEventFrames(). 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.

Because AF 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 PI AF SDK, we can call CaptureValues() on an event frame instance to capture and save attribute values to the AF Server for faster subsequent retrieval. Updating the start or end time of the event frame will automatically recapture the values.

Problem

You have become the Principal Developer at Magical Power Company, and your next task is to develop an energy reporting system for the AF Database "**Magical Power Company**". Please familiarize yourself with this AF Database using PI System Explorer before starting the exercise.

This database contains 12 meters deployed to different buildings within the Wizarding World. Your task is to create AF Event Frames to track average energy usage per day. Each AF Event Frame is based off of an 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()`.

Instructor-Led

1) `AFElementTemplate CreateEventFrameTemplate(AFDatabase database)`

   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. (E.g. "Usage Tracker-Meter002-2016-02-01 00:00:00")

   Create an 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 Hints below for an example configuration string to use.

Student Exercises

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

   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.

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

   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.

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

   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.

See Hints and Examples below for guidance.

Hints

Use the template `NamingPattern` property to define the event frame names. Try setting this property in PI System Explorer first to understand the functionality before using it in PI AF SDK. One possible choice is “%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 ConfigString:

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

This configures the Average Energy Usage attribute to report the average of the referenced element’s Energy Usage over the event frame’s time range.

Examples

Create an AF Event Frame

```
PISystem ps = new PISystems()["AFServer"];
AFDatabase db = ps.Databases["MyDb"];
AFEventFrame ef = new AFEventFrame(db, "MyEF");
AFEventFrame ef = new AFEventFrame(db, "MyEF", efTemp); // from template
```

Create an AF Event Frame Template

```
AFElementTemplate efTemp = db.ElementTemplates.Add("MyEFTemplate");
efTemp.InstanceType = typeof(AFEventFrame);
AFAttributeTemplate pressure = efTemp.AttributeTemplates.Add("Pressure");
pressure.Type = typeof(double);
pressure.DataReferencePlugIn = pressure.PISystem.DataReferencePlugIns["PI Point"];
pressure.ConfigString = @".\Elements[.]|Pressure;TimeRangeMethod=StartTime";
pressure.Description = "Pressure of the tank";
pressure.DefaultUOM = ps.UOMDatabase.UOMClasses["Pressure"].UOMs["pascal"];
```

Set AF Event Frame properties

```
ef.SetStartTime(startTime);
ef.SetEndTime(endTime);
ef.PrimaryReferencedElement = meter;
```

Find Event Frames

```
string query = string.Format("template:\"{0}\" ElementName:\"{1}\"",
       eventFrameTemplate.Name, "Meter003");
AFEventFrameSearch eventframesrch = new AFEventFrameSearch(database,
       "EventFrame Captures", AFSearchMode.StartInclusive, startTime, endTime, query);
foreach (AFEventFrame ef in eventframesrch.FindEventFrames())
{
   // do work
}
```

Capture Event Frame Values

```
string query = string.Format("template:\"{0}\"", eventFrameTemplate.Name);
AFEventFrameSearch eventframesrch = new AFEventFrameSearch(/* omit arguments */);
foreach (AFEventFrame ef in eventframesrch FindEventFrames())
{
       if (!ef.AreValuesCaptured)
              ef.CaptureValues();
}
database.CheckIn();
```

Appendix 1: A Short Introduction to PI AF SDK

Hierarchical Structure

PI AF SDK presents a hierarchical model of objects and collections that represent underlying PI System features and concepts. The `PISystem` is the top-level object used to represent a logical AF Server. Each `PISystem` can have one or more isolated `AFDatabase` objects. Each `AFDatabase` can contain any number of child `AFElement` objects sometimes termed assets. Each `AFElement` object can 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 is ideal for organizing assets which may be grouped in hierarchies and classified using templates created to describe similar objects across the database.

Objects and Collections

Many AF objects have associated collections. For example, each `AFElement` can 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.

PI System Explorer as a Guide

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

Namespaces

PI AF SDK is organized into several namespaces listed below. The most common ones and their basic functionalities are listed below. A block diagram of the namespace organization is available in help.

OSIsoft. AF

The AF namespace contains the main classes used by all the other namespaces within the PI AF SDK. It provides base class implementations and methods for connecting to AF Servers. The primary classes are `PISystem` and `AFDatabase`. Each PI System is composed of any number of isolated databases.

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 enforcement of asset definition. Templates for assets ensure consistency among attributes. Here, you will find `AFElement`, `AFAttribute`, `AFElementTemplates`, etc. and their associated collections (e.g. `AFElements`). In addition, `AFValue`, used to represent a time-series event, resides here.

OSIsoft. AF.Data

The Data namespace provides classes for obtaining data 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.

OSIsoft. AF.PI

The PI namespace provides classes that can be used to manage connections and access information about a **PI Data Archive.** It provides direct access to the PI Data Archive and can be used to find/edit PI Points and read/write data directly to the PI Data Archive.

OSIsoft. AF.Time

The Time namespace provides classes for performing time operations with the PI System. It provides methods for converting time strings including PI Time string syntax (e.g. '*-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.

OSIsoft.AF.EventFrame

The EventFrame namespace provides classes for reading and writing AF Event Frames. These objects can be tied to AF Elements and represent events that occur over a time period, such as equipment downtime or abnormal behavior.

Appendix 2: PI SDK Migration

PI AF SDK data access methods in the OSIsoft.AF.Data namespace and the OSIsoft.AF.PI namespace supersede the functionality provided by PI SDK. Though the PI SDK is still available, it is more appropriate to use PI AF SDK than the PI SDK, except in an architecture that demands only native code. For more information regarding matching PI SDK and AF SDK functionality and migration, please consult the Overview section “PI SDK Equivalents” in the AF SDK online documentation.

In general, PI SDK data method equivalents are found in the OSIsoft.AF.PI namespace. The OSIsoft.AF.Data namespace has equivalent methods with additional functionality. Many new list calls are implemented in PI AF SDK, thus providing potential for performance improvements over PI SDK programs. Also, an AFAttributeList may contain PI, table, and static (constant) data sources aggregated into one result. For the OSIsoft.AF.Data namespace, unit conversions may be requested.

The PI AF SDK also implements parsing for PI time strings similar to that found in PITimeServer. In particular, time arithmetic differences between local and UTC time are handled in the OSIsoft.AF.Time namespace objects. However, AF time objects are not subject to the time zone defined in the ‘localhost.tz’ file defined for each node. The operating system defines the default time zone, which in .NET includes historical time rules similar to those provided in the PI SDK. Time zone settings deviating from the default setting for the node must be explicitly set in the program using the AFTimeZone class, not through the ‘localhost.tz’ file. An additional difference is that the default for AFTime and DateTime .NET objects is to provide fractional seconds for ‘Now’. An explicit method is available to truncate AFTime to integer seconds.

Also, PI digital state sets are returned as AFEnumerationValue objects. AFEnumerationSets expose similar functionality to PI digital state sets with the exception that duplicate state strings cannot be defined in an AFEnumerationSet.

PI buffering functionality for data edits is also available in AF SDK. An additional benefit of AF SDK data write methods is most expose an enumeration to explicitly set buffering behavior on a single call basis. This provides much simpler override functionality if needed.

Appendix 3: Set up Visual Studio for PI AF SDK in a new project

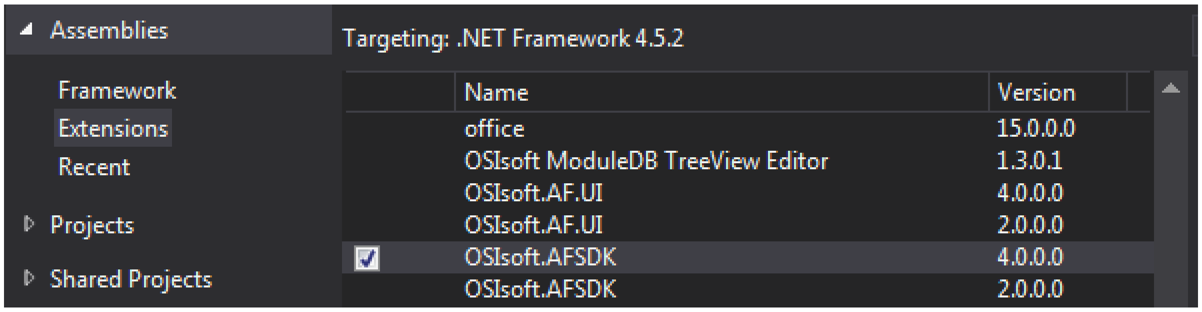
Adding the PI AF SDK Assembly Reference

The Visual Studio Solution for this lab already has a reference to the PI AF SDK added.

However, if you are creating a new Visual Studio Project, you can add a reference to OSIsoft.AF.SDK.dll (4.0.0.0 assembly version) via the following:

1) From the Visual Studio toolbar, click Project>Add Reference.

2) Under Assemblies in the left-pane, choose Extensions, and then select OSIsoft.AFSDK with Version 4.0.0.0.



Targeting: .NET Framework 4.5.2

	Name	Version
	office	15.0.0.0
	OSIsoft ModuleDB TreeView Editor	1.3.0.1
	OSIsoft.AF.UI	4.0.0.0
	OSIsoft.AF.UI	2.0.0.0
✓	OSIsoft.AFSDK	4.0.0.0
	OSIsoft.AFSDK	2.0.0.0

Note: The 4.0.0.0 assembly version is only available when targeting at least .NET Framework 4.x. For exact version requirements, consult the product release notes.

Importing Namespaces

Namespaces can be brought into scope by adding the appropriate `using` directives at the top of the class file.

```
using OSIsoft.AF;

using OSIsoft.AF.Asset;

using OSIsoft.AF.Data;

...
```

Appendix 4: Further Resources

Online PI AF SDK Reference

PI AF SDK Guidelines

PI Developers Club Forums

OSIsoft GitHub Sample Code

vCampus Live 2012 Presentation – PI AF SDK – Performance and Scalability

vCampus Live 2013 Presentation - Optimizing PI AF Server and PI AF SDK Applications

UC 2015 TechCon Hands-On Labs (Unzip and see “Working with the PI AF SDK”)

All of the lab exercises can also be found online on GitHub at

https://github.com/bzshang/PI-AF-SDK-For-Beginners-TechCon-2016

Version Information

Version	Date	Description
1.0	March 14, 2016	Initial version