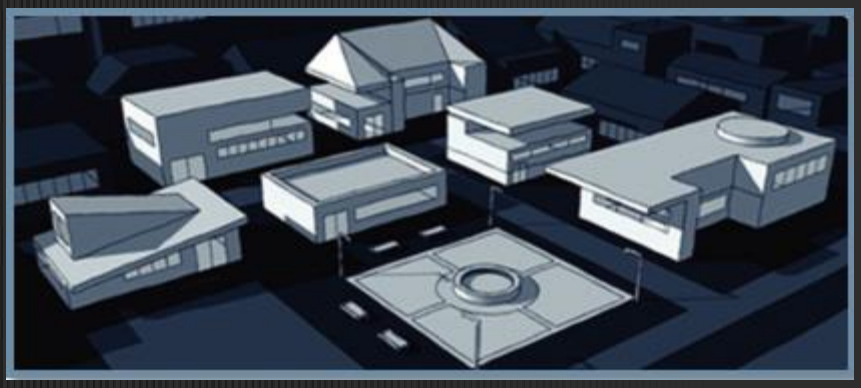


OSIsoft. VIRTUAL CAMPUS

Implementing custom AF 2.x Data References

vCampus White Paper

How to Contact Us

Email: vCampus@osisoft.com
Web: http://vCampus.osisoft.com > Contact Us

OSIsoft, Inc.
777 Davis St., Suite 250
San Leandro, CA 94577 USA

Houston, TX
Johnson City, TN
Mayfield Heights, OH
Phoenix, AZ
Savannah, GA
Seattle, WA
Yardley, PA

Worldwide Offices

OSIsoft Australia
Perth, Australia
Auckland, New Zealand

OSIsoft Europe
Altenstadt, Germany

OSI Software Asia Pte Ltd.
Singapore

OSIsoft Canada ULC
Montreal, Quebec
Calgary, Alberta

OSIsoft, Inc. Representative Office
Shanghai, People's Republic of China

OSIsoft Japan KK
Tokyo, Japan

OSIsoft Mexico S. De R.L. De C.V.
Mexico City, Mexico

Sales Outlets and Distributors

- Brazil
- Middle East/North Africa
- Republic of South Africa
- Russia/Central Asia
- South America/Caribbean
- Southeast Asia
- South Korea
- Taiwan

WWW.OSISOFT.COM

Table of Contents

Overview

About this Document

This document is exclusive to the OSIsoft Virtual Campus (vCampus) and is available on its online Library, located at http://vCampus.osisoft.com/Library/library.aspx.

Any question or comment related to this document should be posted in the appropriate vCampus discussion forum (http://vCampus.osisoft.com/forums) or sent to the vCampus Team at vCampus@osisoft.com.

About this White Paper

Wizards for creating AF 2.x native data references are not yet implemented. With the release of AF 2.6 and Programmed Analytics, many tasks that today require a custom Data Reference will instead be achievable through configuration, therefore the instructions for writing AF 2.x Data References within this document are intended to be **a temporary stop-gap** until that release is ready.

What You Need to Start

You must have the following software installed to develop AF 2.x based custom Data References:

- PI AF 2.x Server and Client
- Microsoft Visual Studio 2010, or 2012
  Note: **Visual Studio 2010 or 2012 and .NET 4.0** is required for implementing data references that supports Rich Data Access (RDA) methods in AF SDK 2.5 and later. Refer to **Addendum: Additional Considerations for AF SDK 2.5 – Rich Data Access**

Implementing AF 2.x Data References

Introduction

Data References in AF 2.x are simply .NET classes in user-created assemblies which derive from the **OSIsoft.AF.Asset.AFDataReference** base class. These assemblies are registered with the AFServer, which stores them into the SQL Server database. When needed by a client application, the assemblies are downloaded to the client and executed by the AFSDK via virtual methods of the AFDataReference class.

Creating your Data Reference Project

Creating Data References in Visual Studio is relatively straightforward. The example below describes creating a C# Data Reference, but other languages – such as VB.NET – can be used as well.

1. In Visual Studio, create a new "**Class Library**" Project.
   (Remember: to be able to support AF SDK 2.5 RDA, you should be using Visual Studio 2010 or 2012 with .NET Framework 4.0)

2. Add a reference to the **OSIsoft.AFSDK** assembly. If you will be providing a "Windows Form" based dialog for configuration, you will also need to add a reference to the **System.Windows.Forms** assembly.

3. Rename the default provided class and file to an appropriate name.

4. Add the appropriate "using" statements. Typically:

```
using System;
using System.ComponentModel;
using System.Globalization;
using System.Runtime.InteropServices;
using System.Text;
using OSIsoft.AF.Asset;
using OSIsoft.AF.UnitsOfMeasure;
using OSIsoft.AF.Time;
```

6. Change your class to inherit from *AFDataReference*:

```
public class StringConcat : AFDataReference
```

7. You **must** provide a Unique ID for your class via the *Guid* attribute. Visual Studio may come with tools to help you generate a new GUID. Go to **Tools → Create GUID**. If you copy the existing samples, you **must** change the Guid otherwise you will overwrite the existing data reference. For example:

```
[Guid("1B2ED32C-98F7-4f1a-86E2-F9EAB5EF0815")]
[Description("String Concat;Concatenate strings.")]
public class StringConcat : AFDataReference
```

7. The *Description* Attribute has the name and description that AF will use, separated by a semicolon. These information are displayed from the AF Server when viewing a list of registered plugins. Use a unique name to differentiate from other plugins.

8. Implement the virtual methods of *AFDataReference*. These are documented in the AF SDK Programming Reference, available as part of the AF Developer Tools from the Download Center. As a minimum, you will likely need to implement the *ConfigString* and *GetValue* methods, as well as the appropriate *SupportedContext* and *SupportedMethods* properties. If your data reference gets input from other attributes then *GetInputs* methods should be implemented Refer to the help and the examples for more details.

9. You can optionally provide a "Windows Forms" based dialog for editing the configuration of your data reference. This is done by creating a Form based dialog, which has a constructor that takes the data reference, and a Boolean read-only flag. This form should then be returned by implementing the *AFDataReference*'s *EditorType* property. Refer to the **IdealGasLaw** example for more details.

   If you do not provide a dialog, then besides the user typing in the configuration string directly, AF will provide a standard **.***NET Properties* window for editing these. In this manner, you can use Microsoft's .NET Component Model for creating a configuration environment.

Registering Data References

The AF 2.0 Client installation contains an executable named **regplugin.exe** used for registering plug-in assemblies. This executable is typically located in the "..\PIPC\AF"

directory. Running the command line ***regplugin /?*** will detail this utility's options, but typically running the following command line ***regplugin <PathToYourAssembly.dll>*** is sufficient. AF 2.1 and higher permit dependent assemblies to be loaded as well. To register a data reference and it's supporting assemblies you can do the following:

Regplugin <PathToYourAssembly.dll>

Regplugin <PathToYourSupportingAssembly.dll> /Owner <YourAssemblyName.dll>

If regplugin.exe has a directory on the command line instead of a single file, it will pick up all DRs in the directory as well as any dependent assemblies.

If you have built your data reference targeting x86 or x64 platforms instead of AnyCPU. You can have the dlls in separate file locations and register them with RegPlugin by referencing the appropriate paths.

IT IS ALSO POSSIBLE TO ADD THE REGPLUGIN STEP TO THE POST-BUILD OF YOUR COMPILE. HOWEVER, IT SHOULD BE NOTED THAT ONCE YOUR PLUGIN IS LOADED INTO AN APPLICATION, IT CANNOT BE UPDATED IN THAT APPLICATION WITHOUT RESTARTING THE APPLICATION.

Debugging Data References

Because the assembly for a data reference is loaded from the AF Database, and not from your compiled output, it is easier to debug by attaching to an existing application (typically the PI System Explorer).

After compiling, register your assembly on the AF Server using **regplugin**.

Copy your .dll and .pdb files to the application data store for AF. This location varies based on operating system, but will be similar to:

```
"C:\ProgramData\OSIsoft\AF\Plugins\<version no of your DR>"
```

And if the plugin is built using RDA and .NET 4.0, it should be copied to:

```
"C:\ProgramData\OSIsoft\AF\Plugins\4.0\<version no of your DR>"
```

Start the application you wish to use to debug (e.g. PI System Explorer), then use Debug/Attach to Process to begin debugging. Note that until the application has a need to use your plugin, it may not yet be loaded. Once you use the data reference from within the application, your breakpoints will become enabled.

Example Data References

This paper references 3 example data references:

- **String Concat** – A simple example of building a string from other attributes' values and strings.
- **Ideal Gas Law** – A simple calculation Data Reference which shipped with AF 1.x and which has been converted to AF 2.0.
- **Rollup** – A Data Reference which traverses one level of the Element Hierarchy computing a roll-up value based on attribute category. Please note that caution should be used with a Data Reference such as this one, as it may need to access a large number of attributes on demand. This can be replaced as a scheduled calculation in upcoming Asset Based Analytics (Abacus).

String Concatenation Data Reference

This String Concatenation Data Reference is a simple example which builds a string by concatenating values of attributes together. It also demonstrates how to use the standard substitution syntax as well as how to look up attribute references.

The String Concatenation Data Reference operates by separating the configuration string into semi-colon separated list of attributes and strings. Strings should be enclosed in double quotes.

The String Concatenation Data Reference does not provide a user interface for editing its configuration string.

Ideal Gas Law Data Reference

The Ideal Gas Law implements a simple calculation of density, given pressure, molecular weight and temperature.

Density = (Pressure * Molecular Weight) / (R * Temperature)

Where R = 0.082057 (atm * L) / (mol * K)

The Ideal Gas Law Data Reference is simple but illustrates some additional capabilities. In particular:

- It provides a form for editing the configuration of the Data Reference;
- It has more sophisticated error handling;
- It is of Unit of Measure aware.

It is similar in structure to AF 1.x wizard based Data References, giving a clearer migration path for anyone who developed 1.x Data References. The Ideal Gas Law Data Reference was shipped as a sample in AF 1.x

Rollup Data Reference

The Rollup Data Reference examines attributes of child elements (one level only) and "rolls" up their value based on simple math calculations (minimum, maximum, average, or sum).

> Note: Because the Rollup Data Reference could traverse many attributes, it must be used with caution. Data References calculate the data they need "on demand". As such, if the Rollup Data Reference were used in an extensive hierarchy of elements, obtaining data at the top level of the hierarchy could be very expensive.

All the examples implement the *GetInputs* method of the Data Reference to inform the AFSDK of the attribute values it requires. By implementing this method rather than retrieving values via *Attribute.GetValue*, the AFSDK is able to much more efficiently retrieve the values. Additionally, it can provide an implementation to *GetValues*, for retrieving historical calculations, as long as one or more of the input attributes come from PI.

Addendum: Additional Considerations for AF SDK 2.5 – Rich Data Access

In AF SDK 2.5, there are some fundamental changes in the library. The AF SDK library now comes with .NET 3.5 and 4.0 versions. The new Rich Data Access (RDA) functions have been added in the .NET 4.0 DLL. Users have access to more data functions for reading data from attributes in a variety of manner. The .NET 3.5 version is still available for backward compatibility, such as for PI ProcessBook where AF related add-ins like AF2 dataset and Element Relative Display only work with the .NET 3.5 library.

How this affects you as a custom data reference developer is that you have to consider the .NET version as you build your custom data reference DLL, and how to have your data reference support the RDA methods. At the time of writing, to be able to support visualization of data from your custom data reference on PI Clients like PI ProcessBook and PI WebParts, it is recommended that you build and register a .NET 3.5 DLL.

When will it make sense for you to build a .NET 4.0 version of your custom data reference? The answer is when you want to leverage the new Rich Data Access (RDA) methods to expose data. Some of the methods, exposed through the AFData object which is introduced in the AF SDK 2.5 (.NET 4.0 version) are:

- RecordedValue
- RecordedValues
- InterpolatedValue
- InterpolatedValues
- InterpolatedValuesAtTimes
- PlotValues
- Summary
- Summaries
- FilteredSummaries
- UpdateValue
- UpdateValues

You may wonder “what happens if I do not attempt to update my data reference to support .NET 4.0 and just have one that works with .NET 3.5?” This basically means that you will be relying on the base implementation of the **AFDataReference** class, which is discussed next.

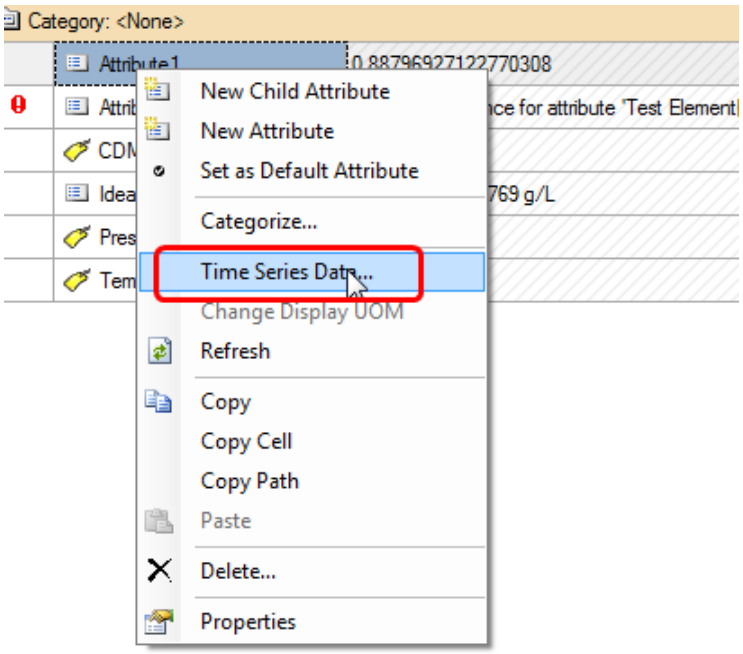
AFDataReference in AF SDK 2.5

If you haven't added support for the RDA methods in the AF SDK within your data reference, the implementation within AFDataReference will determine how your data reference will work.

By default, AFDataReference will indicate that none of the RDA methods are supported (**AFDataMethods.None**). This means that if support for RDA methods is not implemented, any application that calls an RDA method through the **AFData** object will encounter an exception because the method is not supported.

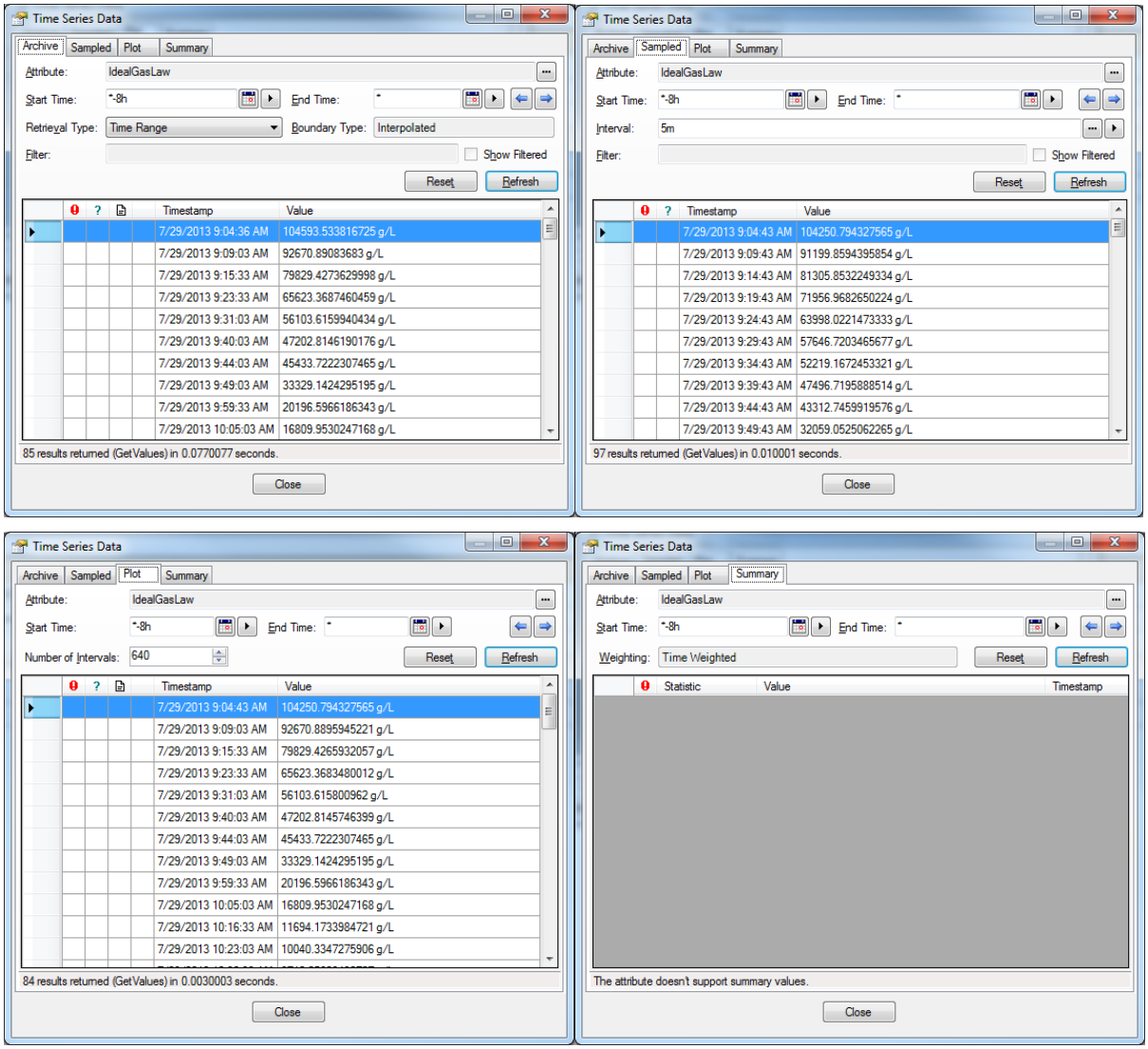
At this point, you may start to wonder something, if you had tried accessing data from your data reference in PI System Explorer via the **Time Series Data** option before. You may have realized that even if you have not added support for RDA methods, data can be retrieved through PI System Explorer, which supposedly uses RDA methods to get data in the Time Series Data dialog window.

To illustrate this, let's look at how an attribute configured to use the Ideal Gas Law data reference (one of the three samples discussed in earlier section) works in PI System Explorer with the **Time Series Data** option.



The Time Series Data option gets archived data for a time period (**AFData.RecordedValues**), interpolated value at a specific interval (**AFData.InterpolatedValues**), plot values (**AFData.PlotValues**) and summary values like average for a time period (**AFData.Summary**).

As you change tabs, data is retrieved from the data reference, without any additional handling or implementation. The only exception is with the Summary tab where no result is received because the method is not supported.



This behavior in PI System Explorer is implemented within PI System Explorer only, and does not represent the behavior of the AFDataReference class. When PI System Explorer sees that the data reference does not support the RDA method, it uses the GetValues method with the appropriate parameters to retrieve values through the data reference instead. Do not assume that the base implementation of AFDataReference class will automatically make use of RDA functionality as you see in PI System Explorer. AFDataReference has a set of base implementations that handle RDA methods appropriately (discussed in the next section), but minimally you will be required to indicate that your data reference supports these RDA methods. (Refer to **Add support for RDA methods**)

Base Implementation of RDA Methods in AFDataReference class

To decide whether to stick to the base implementation of the AFDataReference class or to override the RDA methods with your own implementation, you need to know what the base implementation of AFDataReference is doing. The table below explains the base implementation for each of the RDA methods:

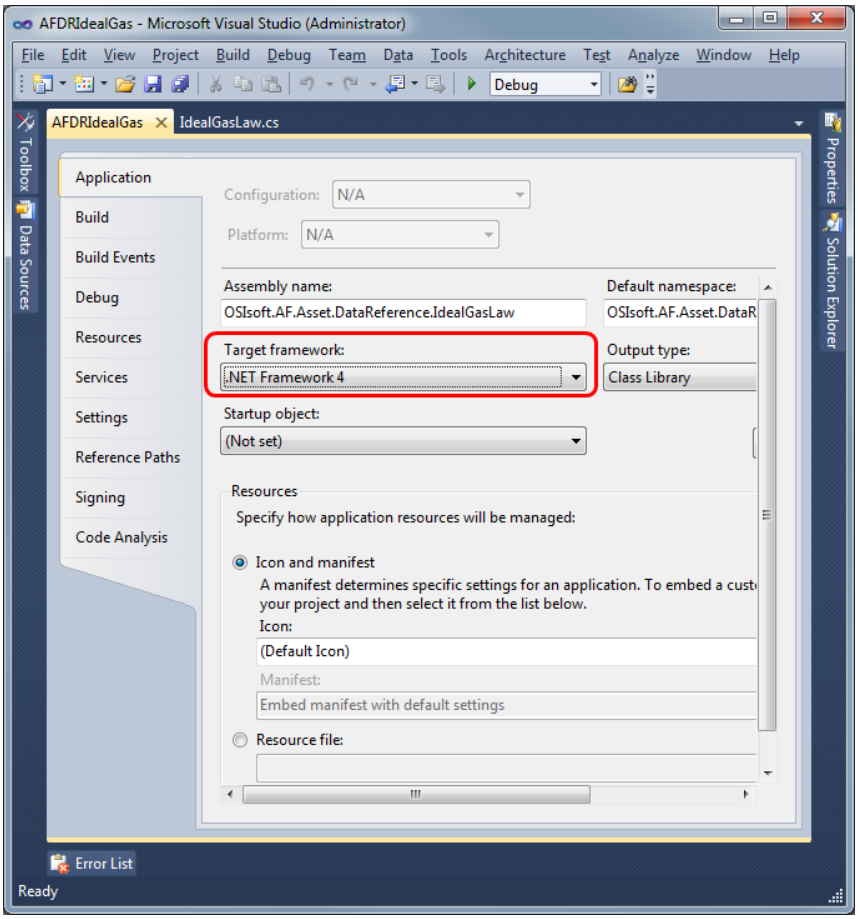
Method	Base Implementation
RecordedValue	Calls **GetValue** to retrieve data at the requested time.
RecordedValues	If the Data Reference calculates values based on other input attributes, then input values and times are used to retrieve data for the time range using **GetValue**. Otherwise, values are generated at the boundaries of the requested time range.
InterpolatedValue	Calls **GetValue** to retrieve data at the requested time.
InterpolatedValues	Calls **GetValue** to retrieve data at each of the timestamps to be sampled.
InterpolatedValuesAtTimes	Calls **GetValue** to retrieve data at the requested timestamps.
PlotValues	If the Data Reference calculates values based on other input attributes, then input times are used to retrieve data for the time range using **GetValue**. Otherwise values are generated at the boundaries of the requested time range.
Summary	Calls **RecordedValues** to retrieve data for the requested timerange, and uses client side summary to get the return value.
Summaries	Calls **RecordedValues** to retrieve data for the requested timerange, and uses client side summary to get the return values.
FilteredSummaries	No implementation, return null.
UpdateValue	No implementation
UpdateValues	No implementaiton

A reminder again, **SupportedDataMethods** property of AFDataReference class will return **SupportedDataMethods.None**. Hence to have the above behavior in your data reference, minimally you would have to override the **SupportedDataMethods** property (see next section).

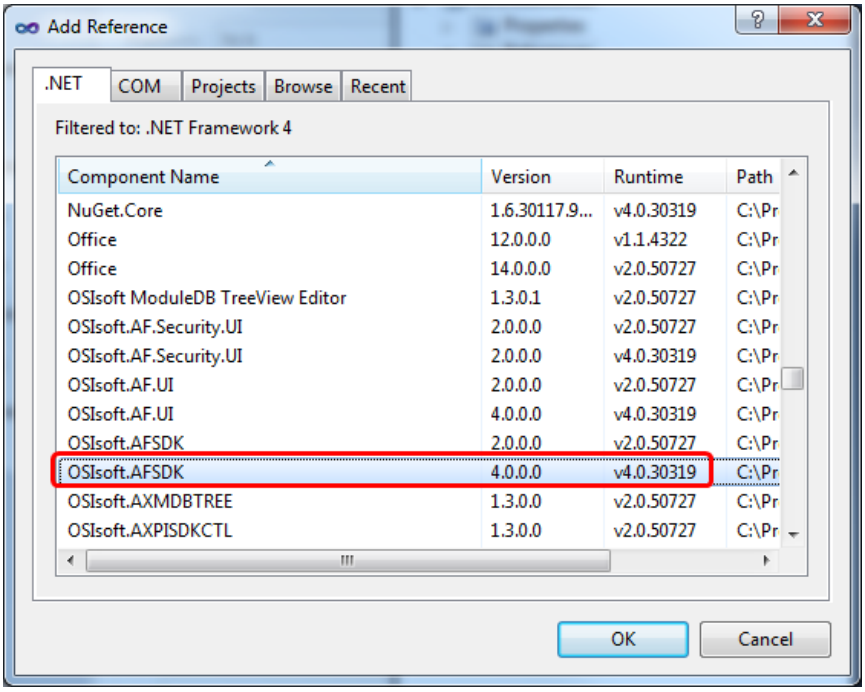
Add support for RDA methods

To add support for RDA methods in your custom data reference,

1. Update your project properties to build against .NET Framework 4.0
   Remember: This is only supported in Visual Studio 2010 or 2012.



2. Reference the .NET 4.0 version of the AF SDK DLL.



3. Override SupportedDataMethods property
   You can choose which of the RDA methods to support by adding the method to support with a “binary or” operation (the pipe operator. “|”). For example:

```
public override OSIsoft.AF.Data.AFDataMethods SupportedDataMethods
{
    get
    {
        return (OSIsoft.AF.Data.AFDataMethods.RecordedValues |
                OSIsoft.AF.Data.AFDataMethods.RecordedValue |
                OSIsoft.AF.Data.AFDataMethods.InterpolatedValue |
                OSIsoft.AF.Data.AFDataMethods.InterpolatedValues |
                OSIsoft.AF.Data.AFDataMethods.InterpolatedValuesAtTimes |
                OSIsoft.AF.Data.AFDataMethods.PlotValues |
                OSIsoft.AF.Data.AFDataMethods.Summary |
                OSIsoft.AF.Data.AFDataMethods.Summaries);
    }
}
```

At this point, your Data Reference will use the base implementation of the RDA methods in the AFDataReference class, when any RDA methods are invoked against an AF attribute using your data reference (refer to the above section). If you are fine, with the base implementation, then you do not have to make any more changes to your data reference.

Of course, there might be cases where you would want to change the behavior, so the calls would work better with your custom data reference. Or you would like to have your own implementations of methods like FilteredSummaries. In these cases, you can choose to:

4. Override the data methods that you want to implement.

```
// utilize GetValue call to retrieve data for RecordedValue
public override AFValue RecordedValue(AFTime time, OSIsoft.AF.Data.AFRetrievalMode
mode, AFAttributeList inputAttributes, AFValues inputValues)
{
    return GetValue(null, time, inputAttributes, inputValues);
}
// utilize GetValues call with numOfValues = 0 to retrieve value
// Assuming you have handled this appropriately in your own GetValues call
public override AFValues RecordedValues(AFTimeRange timeRange,
OSIsoft.AF.Data.AFBoundaryType boundaryType, string filterExpression, bool
includeFilteredValues, AFAttributeList inputAttributes, AFValues[] inputValues,
System.Collections.Generic.List<AFTime> inputTimes, int maxCount = 0)
{
    return GetValues(null, timeRange, 0, inputAttributes, inputValues);
}
// utilize GetValue call to retrieve data for InterpolatedValue
public override AFValue InterpolatedValue(AFTime time, AFAttributeList inputAttributes,
AFValues inputValues)
```

```
{
    return GetValue(null, time, inputAttributes, inputValues);
}


// utilize GetValue call at each of the specified time to retrieve data
public override AFValues
InterpolatedValuesAtTimes(System.Collections.Generic.IList<AFTime> times, string
filterExpression, bool includeFilteredValues, AFAttributeList inputAttributes,
AFValues[] inputValues)
{
    AFValues values = new AFValues();
    int i = 0;
    foreach (AFTime time in times)
    {
        if (inputValues == null)
        {
            AFValue value = InterpolatedValue(time, inputAttributes, null);
            values.Add(value);
        }
        else
        {
            AFValue value = InterpolatedValue(time, inputAttributes, inputValues[i]);
            values.Add(value);
        }
        i++;
    }
    return values;
}
// utilize GetValues call with numOfValues = -ve for InterpolatedValues
// Assuming you have handled this appropriately in your own GetValues call
public override AFValues InterpolatedValuesByCount(AFTimeRange timeRange, int
numberOfValues, string filterExpression, bool includeFilteredValues, AFAttributeList
inputAttributes, AFValues[] inputValues)
{
    return GetValues(null, timeRange, (-1 * numberOfValues), inputAttributes,
inputValues);
}
// utilize GetValues call with numOfValues = +ve for InterpolatedValues
public override AFValues PlotValues(AFTimeRange timeRange, int intervals,
AFAttributeList inputAttributes, AFValues[] inputValues,
System.Collections.Generic.List<AFTime> inputTimes)
{
    return GetValues(null, timeRange, intervals, inputAttributes, inputValues);
}
```

Note that there is no **InterpolatedValues** method for you to override. If you choose to support this method, the method that is called at runtime is the **InterpolatedValuesByCount** method or the **InterpolatedValuesAtTimes** method if the interval is not uniform.

5. Registering and debugging the plugin dll

To use and debug your data reference, you will have to register it to your AF Server. As discussed previous section, **Registering Data References**, this is done using RegPlugins.exe utility.

You can have both .NET 3.5 and 4.0 versions of your custom data reference registered at the same time. These dlls should have different file location. If you follow the convention of the native plugins that come with AF. You can have the .NET 3.5 dlls in**..\PIPC\AF\Plugins** folder while the .NET 4.0 dlls in**..\PIPC\AF\Plugins\4.0** folder.

Given that the GUID of the data reference is the same for both dlls, AF Server will know that these dlls are for the same data reference.

The steps to debug your custom data reference is generally the same as what is described in the previous section, **Debugging Data References**. The application that you use to debug the data reference should allow .NET 4.0 plugins to run, like PI System Explorer.

Revision History

09-Jan-09	Initial draft by Chris Manhard
29-Jan-09	Conversion to vCampus format by Laurent Garrigues
30-Jan-09	Minor edits to text and format by Steve Pilon
1-Aug-13	Addendum added for extending RDA support in AF SDK 2.5
7-Aug-13	Reviewed by John Noss, Paul Kaiser, Ryan Gilbert and vCampus Team after update
19-Aug-13	Revised by after review