Using PI Data with MATLAB

PI DEVELOPERS CLUB WHITE PAPER

OSISOFT, LLC

How to Contact Us

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

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

Table of Contents

Overview

About this Document

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

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

What You Need to Start

You must have the following software installed to use PI data in MATLAB:

- PI Server 3.3. or higher
- MATLAB (the version we tested is R2009a, or 7.8.0.347)
- Either of the following:
    - PI AF Software Development Kit (PI AF SDK)*
    - PI Web API*
    - Microsoft Visual Studio (possibly with PI ACE)
    - PI JDBC Driver + MATLAB's Database Toolbox
    - PI System Management Tools (SMT)
    - PI DataLink
    - The PIConfig utility
    - Microsoft SQL Server with Data Transformation Services (DTS) and Integration Services (SSIS)
    - PI OLEDB Provider
    - PI OPC DA/HDA Server + MATLAB's OPC Toolbox
    - PI OPC DA/HDA Server + Integration Objects' OPC HDA Toolbox

* *Tested with MATLAB version R2014a*

Using PI Data with MATLAB

What is MATLAB?

MATLAB is a language for technical computing that integrates computation, visualization, and programming. It is provided with a graphical environment where one can express problems and get solutions in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning, and allows quickly solving many technical computing problems such as those with matrixes and vectors.

MATLAB features a family of add-on application-specific solutions called toolboxes, some of which we will use in this paper as they allow getting values outside of MATLAB, using standard protocols.

What does MATLAB have to do with PI?

The PI Server and MATLAB are in several ways complementary. For instance, PI is meant to effectively collect and store masses of time series data in real time, whereas MATLAB is in no way a data store. MATLAB and its convenient ad-hoc computing interface extend the computing ability of PI by adding its library of advanced functions such as Fourier transforms, resolution of differential equations, neural networks, Markov chains, etc.

In other words, combining the "real-time data historian" capabilities of PI and the advanced mathematical and computing functions of MATLAB seems like a logical choice in areas like R&D and labs. Another notable advantage of connecting PI to MATLAB is the possibility of continuous MATLAB computations rather than working with a static data sample.

In order to cite some real-world applications of connecting MATLAB to PI, we have prepared some numerical results in the "Simulation" section. We model a PI-based production organization and use the MATLAB optimization toolbox to calculate the optimal production level in real time.

Now we just need to bring that PI data into MATLAB (and most likely push the results of the analyses back to PI).

Ways to Bring PI Data into MATLAB

There are various ways one can employ to make use of PI data in MATLAB – here are the ones we cover in this White Paper.

1. Using PI AF SDK
2. Using PI Web API
3. Programmatically, using a mix of PI and MATLAB functions
4. Using MATLAB's Database toolbox and the PI JDBC Driver
5. Using comma-separated values (CSV) files
6. Using ADO and the PI OLEDB Provider
7. Using MATLAB's OPC Toolbox and the PI OPC DA Server
8. Using Integration Objects' OPC HDA Toolbox and the PI OPC HDA Server

Note that MATLAB is supported on Windows, Linux, Unix and Mac. However, not all of these methods will work on all Operating Systems (OS's). In the above listing, methods #1, 3, 6, 7 and 8 will only work on Microsoft Windows. Method #4 will work on Windows and Linux. Method #2 and 5 will work on all OS's.

Using PI AF SDK

The PI AF Software Development Kit (PI AF SDK) is installed with the PI AF Client and provides programmatic access to PI System data (PI Data Archive and PI AF Server). To access PI System data in MATLAB, we can make use of the rich functionality of PI AF SDK by loading this assembly into the MATLAB environment as a .NET library.

For generic documentation of loading and using .NET library in MATLAB:

- Summary of functions in MATLAB .NET interface (http://www.mathworks.com/help/matlab/ref/net.html)
- Make .NET assembly visible to MATLAB (http://www.mathworks.com/help/matlab/ref/net.addassembly.html)

Programming references for the AFSDK are available in the Tech Support Download Center.

The following examples are tested in MATLAB R2014a and PI AF SDK 2.6. In the following sections, we will be looking into accessing AF data via PI AF SDK.

Loading PI AF SDK into MATLAB and Connecting to the AF Database

First, we will load PI AF SDK as a global .NET assembly into MATLAB and import relevant namespaces:

```
afsdk = NET.addAssembly('OSIsoft.AFSDK');
import OSIsoft.AF.*
import OSIsoft.AF.Asset.*
import OSIsoft.AF.Time.*
import System.*
```

Once PI AF SDK is loaded and imported, we can use its functions to connect to our AF database. We will use the NuGreen database in this example.

```
af_srvs = PISystems;
af_svr = af_srvs.Item('DNG-AF2014');
af_db = af_svr.Databases.Item('NuGreen');
```

Accessing AF Elements

Let's try to get a list of AF elements that start with "B-" and print the element names:

```
af_element_list = AFElement.FindElements(af_db, [], 'B-*', ...
    AFSearchField.Name, true, AFSortField.Name, AFSortOrder.Ascending, ...
    intmax('int32'));

for i = 0:(af_element_list.Count - 1)
    fprintf('%s\n', char(af_element_list.Item(i).Name));
end
```

Getting Values from AF Attributes

The more useful case would be to get values from AF attributes so we can analyze the data in MATLAB. In this example, we will get the current values from the "Process Feedrate" attribute for all elements using the "Boiler" element template.

```
boiler_template = af_db.ElementTemplates.Item('Boiler');
af_attr_list = AFAttribute.FindElementAttributes(af_db, [], '*', [], ...
    boiler_template, AFElementType.Any, 'Process Feedrate', [], ...
    TypeCode.Double, true, AFSortField.Name, AFSortOrder.Ascending, 1000);
af_values = af_attr_list.GetValue();
```

In another example, we will get the compressed values from the "Quality" attribute in the "Houston" element for the past day.

```
plant_template = af_db.ElementTemplates.Item('Plant');
plant_quality = AFAttribute.FindElementAttributes(af_db, [], 'Houston', ...
    [], plant_template, AFElementType.Any, 'Quality', [], ...
    TypeCode.Double, true, AFSortField.Name, AFSortOrder.Ascending, 1000);

time_range = AFTimeRange('*-24h', '*');
my_attribute = plant_quality.Item(0);
af_values = my_attribute.GetValues(time_range, 0, my_attribute.DefaultUOM);
```

To obtain the contents of the `af_values` objects, we can either access the .NET objects directly, or through a .NET helper function, as described below.

I. Access the .NET objects directly

The simplest way to access the data would be to access the .NET objects directly. E.g. to access the first attribute in the list:

```
afattr = af_values.Item(0).Attribute;
afpath = afattr.GetPath();
afval = af_values.Item(0).Value;
aftime = af_values.Item(0).Timestamp;
```

However, this is not sufficient because:

1. For analysis purpose, we would want to deal with the data from PI in the form of array (instead of the AFValues or AFValue object from PI AF SDK).
2. There are some kind of performance overhead when dealing with complex .NET objects like DateTime or AFValue. This can cause performance issues when dealing with large datasets.

II. User a helper function to access the objects

To work around the above issues, we recommend using .NET helper functions to convert the values read through PI AF SDK into simple arrays before handing the data back to MATLAB. MATLAB can then convert the object array into cell array quickly.

.NET helper function (C#)

```
// ...
using OSIsoft.AF.Asset;

namespace OSIsoft.AF.Asset
{
    public class ConvertAFValues
    {
        public static void GetValuesArray(AFValues afvalues, out object[] values,
            out int baddata)
        {
            int size = afvalues.Count;
            values = new object[size];
            baddata = 0;
            int i = 0;
            foreach (AFValue afval in afvalues)
            {
                values[i] = afval.Value;
                if (afval.Status != AFValueStatus.Good)
                    baddata = baddata + 1;
                i = i + 1;
            }
            return;
        }
        public static void GetValuesArray(AFValues afvalues, out object[] values,
            out double[] timestamps, out int[] statuses, out int baddata)
        {
            int size = afvalues.Count;
            values = new object[size];
            timestamps = new double[size];
            statuses = new int[size];
            baddata = 0;
            int i = 0;
            foreach (AFValue afval in afvalues)
            {
```

```
                values[i] = afval.Value;
                timestamps[i] = ((DateTime)afval.Timestamp).ToOADate() + 693960;
                statuses[i] = (int)afval.Status;
                if (afval.Status != AFValueStatus.Good)
                    baddata = baddata + 1;
                i = i + 1;
            }
            return;
        }
    }
}
```

Note that the timestamps are incremented by 693960 to convert the serial date number from Ole Automation Date (ConvertAFValues used the .NET method ToOADate()) to the serial date number used in MATLAB since the base date used is different. In MATLAB, a serial date number of 1.0 represents January 1, 0000, whereas Ole Automation date of 1.0 represents 31 December 1899. We have therefore included the offset of 693960 to account for the number of days between the two base dates.

We made the Visual Studio project (**ConvertAFValues.zip**) available for download in PI Developers Club on PI Square.

To use the helper function, we have to add the .NET assembly into MATLAB:

```
convert = NET.addAssembly('C:\Users\dng\Documents\ConvertAFValues.dll');
```

We can then get data by using the helper function and converting the data into MATLAB cell arrays:

```
[values, times, status, badflags] =
   ConvertAFValues.GetValuesArray(af_values);
```

Or `[values, badflags] = ConvertAFValues.GetValuesArray(af_values);`

```
mlvalues = cell(values);
mltimestamps = double(times);
mlstatus = int32(status);
```

Once the values are in MATLAB cell arrays, various data manipulations can be performed (make sure to check the health and data types of the values before performing data manipulations). E.g.

```
if badflags == 0
    mean_value = mean(cell2mat(mlvalues));
    ts = timeseries(cell2mat(mlvalues),mltimestamps);
end
```

Writing back to AF

Finally, we might want to write the calculated data back into AF attributes. In this example, we will write a random value to the “Energy Savings” attribute in the “Houston” element:

```
plant_EnergySavings = AFAttribute.FindElementAttributes(af_db, [], ...
    'Houston', [], plant_template, AFElementType.Any, 'Energy Savings', ...
    [], TypeCode.Double, true, AFSortField.Name, AFSortOrder.Ascending, ...
    1000);
afitem = plant_EnergySavings.Item(0);
new_afval = AFValue(afitem, rand*1000, AFTime.Now);
afitem.SetValue(new_afval);
```

Useful MATLAB Functions

We have created a set of .m files which makes use of PI AF SDK functions in MATLAB:

- **mAF_GetElementsFromTemplate**: Returns all the Elements based on a specified Element Template.
- **mAF_GetAttributesFromElement**: Returns all the attributes from a specified Element.
- **mAF_GetAttributeSnapshot**: Returns the snapshot value of a specified Attribute
- **mAF_GetAttributeArchiveValues**: Returns the archived value of a specified Attribute at the certain timestamp.
- **mAF_GetAttributeInterpolatedValues**: Returns a series of sampled (interpolated) values for the specified Attribute in the specified time range, at the specified interval
- **mAF_GetAttributeAverage**: Returns the time weighted average for a specified attribute in a specified time range.
- **mAF_UpdateAttributeValue**: Writes a random value in the specified attribute with the current timestamp

These functions are available for download as **AFSDK_MATLAB.zip**.

Using PI Web API

PI Web API is a RESTful PI System access layer that provides a cross-platform programmatic interface to the PI System. Programming references are available at the PI Web API's online help, available by default at https://<yourWebAPIserver>/piwebapi/help.

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

The following examples are tested with MATLAB R2014a and PI Web API 1.2.0.

Working with JSON

We will first add the MATLAB toolbox JSONlab to the MATLAB search path:

```
addpath('C:\Users\dng\Downloads\jsonlab_1.0beta\jsonlab');
```

We will then connect to our NuGreen AF database and download its content into a MATLAB string by using the MATLAB function urlread. We will then parse the JSON string into a structure array by the loadjson function (from JSONlab).

```
base_url = 'https://dng-web.osisoft.int/piwebapi';
af_svr_name = 'DNG-AF2014';
af_db_name = 'NuGreen';

afdb_url = strcat(base_url, '/assetdatabases?path=\\', af_svr_name, ...
    '\', af_db_name);
afdb_json = loadjson(urlread(afdb_url));
```

afdb_json is a 1x1 struct with the following content:

```
>> afdb_json

afdb_json =

          WebId: 'D0G1eh97OAw0KXIEzJrhPzTAbg2lcKh3qE613Io88dDAPARE5HLUFGMjAxNFxOVUdSRUVO'
             Id: '70a50d6e-77a8-4ea8-b5dc-8a3cf1d0c03c'
           Name: 'NuGreen'
    Description: 'PI BI Project Asset Model'
           Path: '\\DNG-AF2014\NuGreen'
          Links: [1x1 struct]
```

We can then access various information in the structure array by using the dot notation of the form `structName.fieldName`, e.g. `afdb_json.Name`, `afdb_json.Links.Self`, etc.

Note that if you are using a development environment and does not have a valid SSL certificate for PI Web API, you might get an error using MATLAB's urlread function. You will need to first import the untrusted certificate into the MATLAB's JRE keystore by following the procedures at http://www.mathworks.com/matlabcentral/answers/92506-can-i-force-urlread-and-other-matlab-functions-which-access-internet-websites-to-open-secure-websi. Alternatively, you can import the certificate manually by navigating to the MATLAB folder (e.g. C:\Program Files\MATLAB\R2014a\sys\java\jre\win64\jre\bin) and importing the certificate. E.g.

```
keytool -import -file "C:\Projects\Matlab with Web API\dng_code.cer" -alias
      dngcode -keystore "C:\Program Files\MATLAB\R2014a\sys\java\jre\win64\
      jre\lib\security\cacerts.org" -storepass changeit
```

Accessing AF Elements

We will again try to get a list of AF elements that start with “B-”and print the element names. We will search the elements by the name filter “B-*” and include the “searchFullHierarchy” flag to return all elements in the database.

```
asset_element_url = afdb_json.Links.Elements;
asset_element_json = loadjson(urlread(asset_element_url));
element_base_url = asset_element_json.Items{1}.Links.Elements;

element_filter = 'B-*';
element_url = strcat(element_base_url, '?nameFilter=',element_filter, ...
    '&searchFullHierarchy=true');
element_json = loadjson(urlread(element_url));

for i = 1:length(element_json.Items)
    fprintf('%s\n', element_json.Items{i}.Name);
end
```

Getting Values from AF Attributes

JSON’s simple data types can be converted and used by MATLAB directly without much overhead (as opposed to .NET objects in PI AF SDK). We will see in the following example how easy it is to work with data returned by PI Web API. We will be getting the compressed values from the “Quality” attribute in the “Houston” element for the past 8 hours.

Since we are looking at a specific attribute, we will find data for the attribute by searching for the `Houston|Quality` attribute by its full path:

```
attribute_path = strcat('\\', af_svr_name, '\', af_db_name, ...
    '\NuGreen\Houston|Quality');
attribute_url = strcat(base_url, '/attributes?path=', attribute_path);
attribute_json = loadjson(urlread(attribute_url));
```

The `attribute_json` structure contains the link for the compressed value (represented by “RecordedData”) where we can access by `attribute_json.Links.RecordedData`. From there, we will specify the start and end time by including them in the query string:

```
start_time = '*-8h';
end_time = '*';
data_url = strcat(attribute_json.Links.RecordedData, '?startTime=', ...
    start_time, '&endTime=', end_time);
data_json = loadjson(urlread(data_url));
```

Finally, we can put the values and timestamps directly into MATLAB arrays and cell array, respectively.

```
mlvalues = zeros(1,length(data_json.Items));
mltimestamps = cell(1,length(data_json.Items));
for k = 1:length(data_json.Items)
    mlvalues(k) = data_json.Items{k}.Value;
```

```
    mltimestamps{k} = data_json.Items{k}.Timestamp;
end
```

Writing back to AF

To write values back into AF, we will need to do a PUT or POST to send the JSON object to the PI Web API server. We will start by finding the URL where we will PUT/POST our data to. In this example, we will write to all elements that are using the "Process" template.

```
process_template = 'Process';
processT_url = strcat(element_base_url, '?templateName=', ...
    process_template, '&searchFullHierarchy=true');
processT_json = loadjson(urlread(processT_url));
```

For each element (indexed by i), we will look for the "Write Test PUT" attribute (this attribute was added to the AF database NuGreen beforehand):

```
write1_path = strcat(processT_json.Items{i}.Path, '|Write Test PUT');
write1_url = strcat(base_url, '/attributes?path=', ...
    strrep(write1_path, ' ', '%20'));
write1_json = loadjson(urlread(write1_url));
write1_value_url = write1_json.Links.Value;
```

On the `write1_value_url` page, we can pull up the documentation for the PUT method at https://<yourWebAPIserver >/piwebapi/help/PUT-attributes-webId-value. This call is intended for use with attributes that have no data references, which is true for our "Write Test PUT" attribute (data reference: None).

To PUT JSON data on the `write1_value_url` page, we created the WriteValues class with the JSONWrite function in Java (JDK 1.7). The Java NetBeans project is available for download as **WebAPIwrite.zip**.

```
public static int JSONWrite(String putPostOption, String address, String
jsonObj)
```

The JSONWrite function takes the following parameters:

1. PUT or POST method: specify whether the PUT or POST method should be used (formatted as String "PUT" or "POST").
2. URL whether the JSON data will be put/posted: In this example, the web address is contained in `write1_value_url`.
3. JSON object containing the value: construct the JSON object in the String format.

The function returns the response code in integer. Information corresponding to each response code can be found on the PI Web API's online help page.

To use the Java WriteValues class, we will first add references to the class:

```
javaaddpath('C:\Users\dng\Document\WebAPIWrite\build\classes')
ww = dng.webAPI.WriteValues;
```

We will then construct and PUT our JSON string into `write1_value_url`:

```
write1_value_string = '{"Value": 30}';
write1_response = ww.JSONWrite('PUT', write1_value_url,
write1_value_string);
```

We have just wrote the value of 30 in the “Write Test PUT” attribute!

Similarly, we can construct and POST a JSON string into another attribute which has a PI Point data reference (refer to https://<yourWebAPIserver>/piwebapi/help/POST-attributes-webId-attributes for more information).

```
write2_value_string = '{"Timestamp": "2014-08-22T01:00:00Z", "Value": 40}';
write2_response = ww.JSONWrite('POST', write2_value_url, ...
        write2_value_string);
```

Programmatically, in Languages Like VB6, VB.NET and C#

By adding a reference to the MATLAB COM Automation Server to your projects, you can invoke MATLAB functions from your own applications. While most COM-based and .NET languages will support this, this section demonstrates a simple example in VB6, VB.NET and C#.

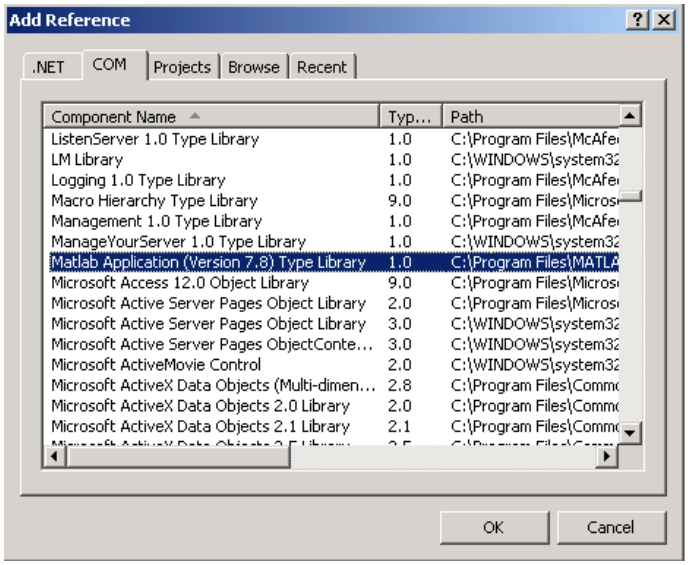
Please understand **we are by no means MATLAB experts** (we are PI experts ☺) and therefore we invite you to use the following resources for more information and assistance on programming with MATLAB:

- In the MATLAB help: "MATLAB COM Automation server Support" (under the "External Interfaces" section)
- MathWorks Technical Support (http://www.mathworks.com/support/)
- MathWorks Discussion Forums (http://www.mathworks.com/matlabcentral/newsreader/)

With that said, let's now code!

Adding a Reference to MATLAB

We first need to add a reference to **Matlab Application (Version x.y) Type Library**
(**note 1:** for .NET users, this is a COM reference)
(**note 2:** that it might also show up as "Matlab Automation Server Type Library")



The MATLAB Code

The code to execute MATLAB functions is relatively simple - in a nutshell, you need to:

a) Declare an object and initialize it as an MLAPP.MLAPP object
b) Send some data to MATLAB
c) Execute functions in MATLAB
d) Retrieve the computed data from MATLAB
e) Use the data contained in array oOutput in your code as you would use any other cell array.
   Here are code examples that demonstrate this:

VB6

```
Dim oMatlab As Variant
Set oMatlab = New MLApp.MLApp
sResult = oMatlab.PutWorkspaceData("x", "base", dblInput)
sResult = oMatlab.Execute("y = MyFunction(x);")
sResult = oMatlab.GetWorkspaceData("y", "base", oOutput)
```

VB.NET

```
Dim oMatlab As New MLApp.MLApp()
oMatlab.PutWorkspaceData("x", "base", dblInput)
oMatlab.Execute("y = MyFunction(x);")
oMatlab.GetWorkspaceData("y", "base", oOutput)
```

C#

```
MLApp.MLAppClass oMatlab = new MLApp.MLAppClass();
oMatlab.PutWorkspaceData("x", "base", dblInput);
oMatlab.Execute("y = MyFunction(x);");
oMatlab.GetWorkspaceData("y", "base", out oOutput);
```

In these examples, *dblInput* is declared as a 1-dimension array of type double (a 64-bit floating point number) and *oOutput* is declared as an Object (a Variant, in VB6).

Sending Collections of PI Values to MATLAB

Whether you are using this within a PI ACE module, in an ADO/ADO.NET project or in a custom application using PI SDK or PI AF SDK, chances are you will need to send a collection of values to MATLAB. This means that at some point, you will need to convert the object that holds the PI data to an array of numbers (e.g. the *dblInput* variable in the examples above).

Examples of these objects are:

- A **PIValues** collection in PI ACE or PI SDK
- An **AFValues** collection in PI AF SDK
- A **DataSet** object in ADO.NET (or a Recordset object in ADO)

Consequently, you should consider developing a simple a function that takes one of these objects as a parameter, loops through its elements, and copies its values into an array of numbers it will return.

The actual retrieval of the PI values (e.g. using PI SDK, PI OLEDB, etc.) is beyond the scope of this document. We invite you to ask your questions on the PI Developers Club Discussion Forums and look at the various documents available on the Tech Support Download Center.

In the "Simulation" section we have used this programmatic approach to build a model of a three-site production enterprise. We assume that the real-time price variables are stored in PI and use MATLAB to calculate and write back to PI the profit-maximizing production levels for each site.

Using this in PI ACE

In case you want to use this in PI ACE, here are a few recommendations:

1. **Declare** the MATLAB Application object (MLAPP.MLAPP) at the class level
   (in VB6, declare it as Variant in the "User Variable Declarations" section)
2. **Initialize** that application object in the "ModuleDependentInitialization" routine
   (in VB6, that corresponds to the "UserDefinedInitialization" routine)
3. **Free up** that application object in the "ModuleDependentTermination" routine
   (in VB6, that corresponds to the "UserDefinedTermination" routine)
4. **Perform** your calculation logic in the "ACECalculations" routine
   (in VB6, that corresponds to the "ActualPerformanceEquations")

Here are examples of you the code would look like:

VB6

```
''''''            User Variable Declarations            ''''''
Private oMatlab As Variant

' ...

Public Sub ActualPerformanceEquations()
    Dim sResult As String
```

```
    Dim oOutput As Variant
    ' ...
    sResult = oMatlab.PutWorkspaceData("x", "base", cdt158.Value)
    sResult = oMatlab.Execute("y = MyFunction(x);")
    sResult = oMatlab.GetWorkspaceData("y", "base", oOutput)
    ' ...
    sinusoid.Value = oOutput
End Sub

Private Sub UserDefinedInitialization()
   Set oMatlab = New MLApp.MLApp
End Sub

Private Sub UserDefinedTermination()
   Set oMatlab = Nothing
End Sub
```

VB.NET

```
Public Class MyFunction
    Inherits PIACENetClassModule
    Private inputPoint As PIACEPoint
    Private outputPoint As PIACEPoint
    Private oMatlab As MLApp.MLApp

    ' ...

    Public Overrides Sub ACECalculations()
        Dim oOutput As Object
        ' ...
        oMatlab.PutWorkspaceData("x", "base", inputPoint.Value)
        oMatlab.Execute("y = MyFunction(x);")
        oMatlab.GetWorkspaceData("y", "base", oOutput)
        ' ...
        outputPoint.Value = oOutput
    End Sub

    ' ...

    Protected Overrides Sub ModuleDependentInitialization()
        oMatlab = New MLApp.MLApp()
    End Sub

    Protected Overrides Sub ModuleDependentTermination()
        oMatlab = Nothing
    End Sub
End Class
```

C#

By default, PI ACE does not support C# - but that doesn't mean you can't do it ☺

You need to use a combination of the code shown previously and the concepts shared in the **Writing a PI ACE Calculation in C#** tutorial.

Using MATLAB's Database Toolbox and the PI JDBC Driver

With this approach you will use the **MATLAB Database Toolbox** (an optional MATLAB software component) and the Java Database Connectivity (JDBC) standard to obtain data from PI. The PI products required for this are the **PI JDBC Driver**, the PI SQL Data Access Server (**PI SQL DAS**) and the **PI OLEDB Provider**, all available on the Tech Support Download Center.

The PI JDBC Driver is an implementation of the JDBC standard based on the JDBC 4.0 API (Java Platform SE 6). PI JDBC is a type 1 JDBC driver (bridge) that employs the PI OLEDB Provider to connect to the PI Server and execute queries. Communication from PI JDBC to PI OLEDB requires using the PI SQL Data Access Server (PI SQL DAS), which serves as a gateway between PI JDBC and PI OLEDB. It provides secure network communication (https) to PI JDBC and executes queries as a PI OLEDB consumer (client). PI OLEDB and PI SQL DAS run on Windows whereas PI JDBC is supported on Windows and Linux operating systems.

To connect to PI using the MATLAB Database Toolbox and the PI JDBC Driver, follow these steps:

1. Download and install the MATLAB Database Toolbox

2. Download and install the PI JDBC Driver on the computer where MATLAB is installed (the PI SQL DAS can be installed on another machine)

3. Close MATLAB if it is running

4. Insert the full path to the JAR file (the actual JDBC driver) to the "classpath.txt" file, located at $MATLABROOT\toolbox\local\classpath.txt
   (by default: C:\Program Files\PIPC\JDBC\PIJDBCDriver.jar)

5. Start MATLAB

6. Type the following commands in the Command window.
   Note that these commands could be wrapped into .m files (MATLAB code files) like the PI OLEDB, the OPC DA and the CSV examples contained in this White Paper.

   a. This command (all on 1 line) instructs PI JDBC to establish a connection to the specified PI Server (`PIServer`) via the specified PI SQL DAS Server (`PISQLDASServer`).

```
conn = database
('PI','','','com.osisoft.jdbc.Driver',
 'jdbc:pisql://PISQLDASServer/Data Source=PIServer;
 Integrated Security=SSPI')
```

b. Now we retrieve data from PI using a SQL query. You can consult the PI OLEDB Provider Documentation (available on the Tech Support Download Center) for more information on the SQL queries it possible to execute.
The second command, `fetch`, imports the rows of data from the open SQL cursor `curs` into a MATLAB cell array, structure or numeric matrix.

```
curs = exec(conn, 'SELECT tag, descriptor FROM PIPoint.Classic')
curs = fetch(curs)
```

c. You can now make use of the data contained in the cursor object to perform various MATLAB operations. This is done via `curs.data`, which consists of a cell array by default.

d. When you are done, close the cursor and the connection.

```
close(curs)
close(conn)
```

USING CSV FILES

Comma-Separated Values (CSV) files are plain text files that prove very useful for sharing information; they contain one record per line and the various elements of that record are separated by commas. This part of the paper is not limited to PI (you could get any kind of data into a CSV file) but we decided to provide an example MATLAB function that handles parsing of CSV files into cell arrays.

For example, one could get the following PI data exported to a CSV file:
(where 'cdt158' is one of the simulation PI Points created by default)

```
cdt158, 09-aug-09 11:00:00, 123
cdt158, 09-aug-09 12:00:00, 234
cdt158, 09-aug-09 13:00:00, 345
```

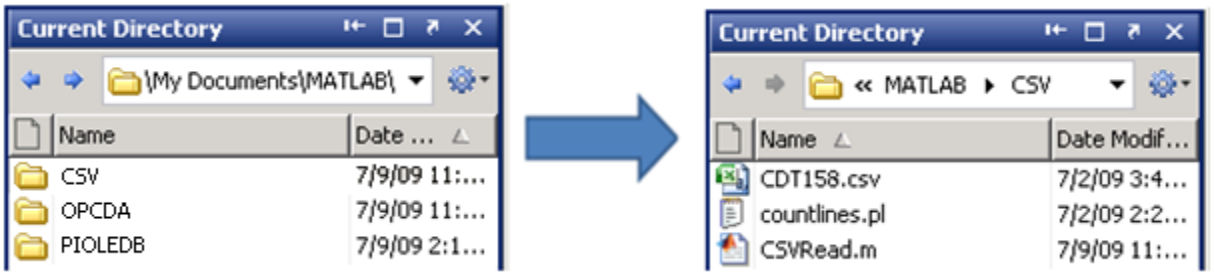
And have our MATLAB function turn it into a cell array for further manipulation:

cdt158	09-aug-09 11:00:00	123
cdt158	09-aug-09 12:00:00	234
cdt158	09-aug-09 13:00:00	345

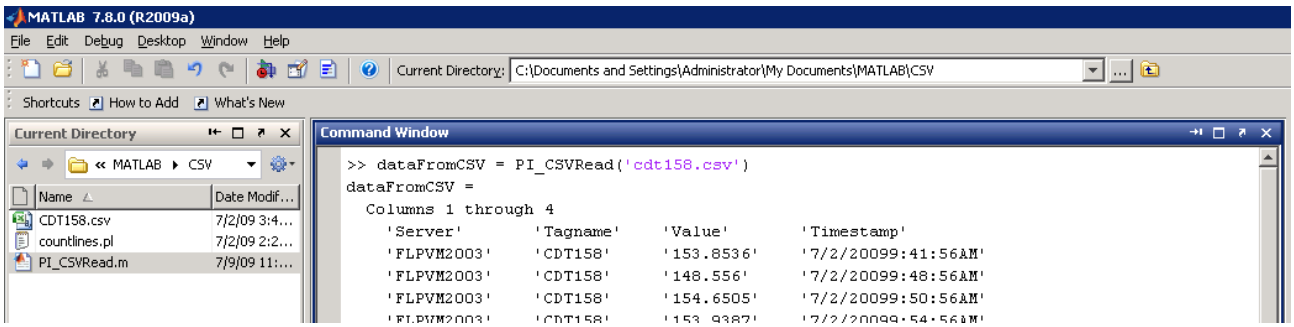
Before we describe a few different ways to create CSV files with PI data, let's see how you can use the "CSV file reading" MATLAB function we created:

1. Download the package (we made it available as a "PItoMATLAB.zip")
2. Extract the content of the "CSV" folder in the .zip file, to the directory of your choice (e.g. My Documents\MATLAB\CSV)

3. Open MATLAB and set this directory as the "current directory"



4. Test the **CSVRead** function in the Command window with the sample **cdt158.csv** file provided:



The **CSVRead** function is designed to read CSV files of any number of rows and any number of columns. You can make use of this function in your MATLAB programs to import any kind of data stored in CSV files; how you manipulate the cell array and what you do with that information in other MATLAB functions, is then up to you.

How to get a CSV file filled with PI Data

There are a few different ways to get PI data saved into a CSV file – just to name a few:

- **Using PI System Management Tools (SMT)**
  a. Open SMT
  b. Go to the *Recorded Values* under the *Data* section
  c. Get data for a certain PI Point
  d. Use the *Export events to a CSV file* tool in the toolbar

- **Using Microsoft Excel with the PI DataLink add-in**
  a. Open Excel
  b. Make sure the A1 cell is selected
  c. Get data on the spreadsheet using PI DataLink functions (e.g. *Compressed Data*, *Sampled Data*, etc.)
  d. Save As CSV

- **Using the PIConfig utility**
  a. Save the following piconfig instructions into a file named **instructions.txt**:

```
@table piarc
@istru tag, starttime, endtime, count, mode
@ostr time, value
@ostr ...
cdt158,y,t,86401,even
```

  b. From a command prompt, execute the following command to obtain a file named **outputValues.m** (execute it in the folder where **instructions.txt** was saved)

```
%PISERVER%\adm\piconfig < instructions.txt > outputValues.csv
```

  c. Clean the file to keep only values. For instance, the first line and the last six (6) lines should be deleted as they represent piconfig logging.

- **Using some Microsoft SQL Server tools and PI OLEDB Provider**
  You can extract data from Microsoft SQL Server and generate CSV files using the PI OLEDB Provider some Microsoft SQL Server tools: Data Transformation Services (DTS) and SQL Server Integration Services (SSIS).

  This is described in the White Paper named "Utilizing MS SQL Server Tools (DTS and SSIS) for Data Exchange between PI and RDBMS".

Using ADO and the PI OLEDB Provider

This approach makes use of the ADO objects contained in the Microsoft Data Access Components (MDAC, a default component of Microsoft Windows since Microsoft Windows 2000) and the PI OLEDB Provider, which provides SQL-based access to the PI Server. It requires that the PI OLEDB Provider be installed on the same machine as MATLAB.

MATLAB provides functions such as **actxcreate**, **invoke**, **get** and **set** to interact with COM objects such as the ADO objects. While we could have created our own ADO wrapper in MATLAB, we based our work on a package developed by Martin Frulan (martin.furlan@iskra-ae.com) that does it well: it's a series of .m files (MATLAB code files) which provide functions that handle the main ADO objects and methods we will need. These functions are:

- **adodbcn**: Create, connects and returns an ADO connection given a connection string
- **adodbcnstr:** Creates and returns a connection string given the passed parameters. This function handles connections to Microsoft Access, MS SQL Server and Oracle.

Note: We did not make use of this function to build our own "PI" connection string; you could however modify this function to have it handle the "PI" type of connection.

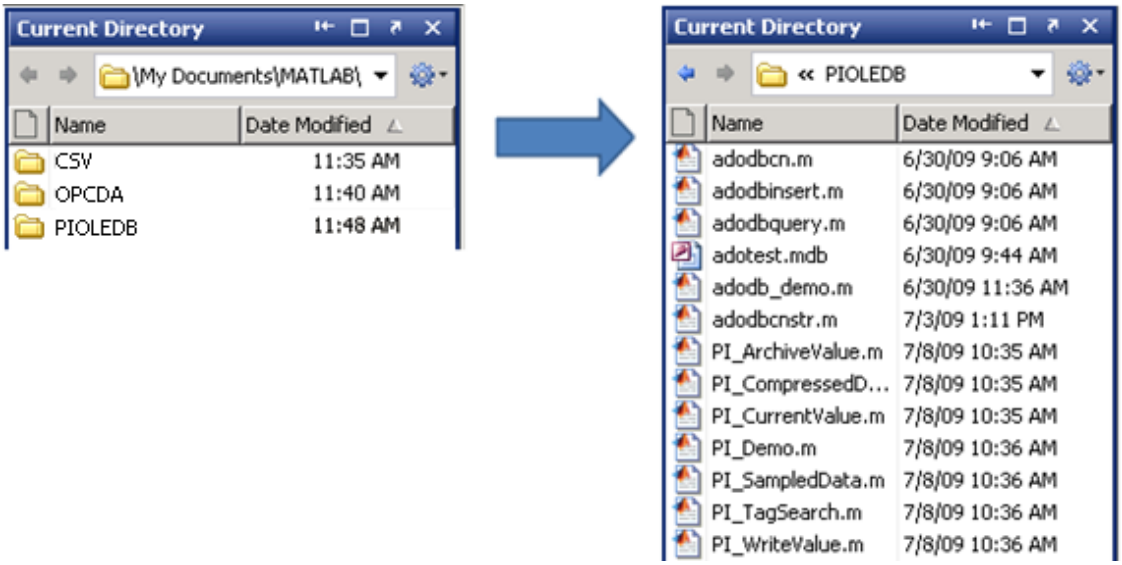
- **adodbquery:** Execute the specified SQL query against the specified ADO Connection and returns the results
- **adodbinsert**: Same as above, but meant for queries that do not return a record set (e.g. INSERT, UPDATE and DELETE queries)
- **adodb_demo**: Some demo code that makes use of the above functions (a sample Microsoft Access database named adotest.mdb is provided)

In order to show how to make use of this and the PI OLEDB Provider, we created our own set of .m files which consist of PI-specific functions that make use of the above ones:

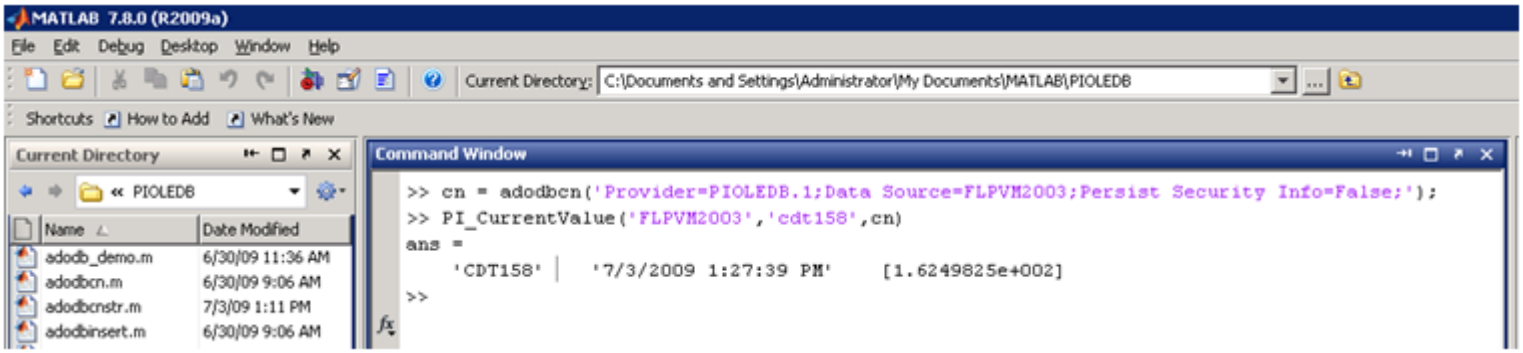
- **PI_CurrentValue**: Returns the snapshot value for the specified tag
- **PI_ArchiveValue**: Returns the archive value at the specified timestamp for the specified tag
- **PI_CompressedData**: Returns a series of compressed (archived) values for the specified tag in the specified time range
- **PI_SampledData**: Returns a series of sampled (interpolated) values for the specified tag in the specified time range, at the specified interval
- **PI_WriteValue**: Writes the specified value in the specified tag at the specified timestamp
- **PI_TagSearch**: Returns the list of tags that correspond to the specified tag mask
- **PI_Demo**: Some demo code that makes use of the above functions (make sure you edit the name of the PI Server to a name that is in the PI SDK Known Servers List)

To make use of the above functions in MATLAB:

1. Download the package (we made it available as "PItoMATLAB.zip")
2. Extract the content of the "PIOLEDB" folder in the .zip file, to the directory of your choice (e.g. My Documents\MATLAB\PIOLEDB)
3. Open MATLAB and set this directory as the "current directory"



4. Test the **PI_ArchiveValue**, **PI_CompressedData**, **PI_CurrentValue**,**PI_SampledData**, **PI_TagSearch** and **PI_WriteValue** functions in the Command window (a sample command is provided in each .m file).



Also note that the **PI_Demo.m** file contains a script that tests each one of these functions and show you how you can make use of these functions in your MATLAB programs.

If you are familiar with ADO (e.g. in VB6, VBA, VBScript, JavaScript), you might want to write your own ADO wrapper in MATLAB rather than use the one we used. Below is the correspondence between some lines of MATLAB code and their Visual Basic equivalent.

MATLAB	***Visual Basic***

`% Connection`
`cn =`
`actxserver('ADODB.Connection');`
`set(cn,'CursorLocation',3);`
`invoke(cn,'Open', cnstr);`
`set(cn,'CommandTimeout',60);`
	`' Connection`
	`Dim cn As ADODB.Connection`
	`cn.CursorLocation = 3 '`
	`adUseClient`
	`cn.Open cnstr`
	`cn.CommandTimeout = 60`

`% Query/Transaction`
`invoke(cn,'BeginTrans');`
`r = invoke(cn,'Execute',sql);`
`invoke(cn,'CommitTrans');`
`% or invoke(cn,'CommitTrans');`
	`' Query/Transaction`
	`cn.BeginTrans`
	`Set r = cn.Execute(sql)`
	`cn.CommitTrans`
	`' or cn.RollbackTrans`

`% Recordset`
`x=invoke(r,'getrows');`
`invoke(r,'close');`
	`' Recordset`
	`Set x = r.GetRows`
	`r.Close`

`% Closing`
`invoke(cn,'close');`
`invoke(r,'close');`
	`' Closing`
	`r.Close`
	`cn.Close`

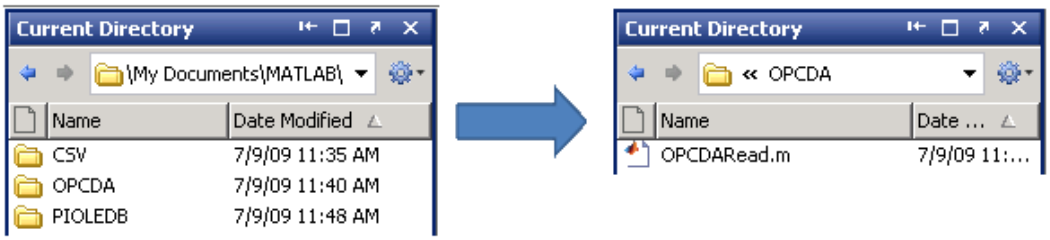
`% Releasing`
`clear cn r;`
	`' Closing`
	`Set r = Nothing`
	`Set cn = Nothing`

Using MATLAB's OPC toolbox and the PI OPC DA Server

This approach uses the OPC DA specification (OLE for Process Control, Data Access) to bring PI data into MATLAB, which allows reading and writing real-time data from OPC DA servers (i.e. current values, no history).

The key to this approach is the use of the PI OPC DA/HDA server (its DA portion) and the MATLAB OPC Toolbox. This toolbox is an add-on to MATLAB and provides a library of function for connectivity between any OPC DA server and MATLAB.

1. Download and install the MATLAB OPC Toolbox
2. Download the package (we made it available as "PItoMATLAB.zip")
3. Extract the content of the "OPCDA" folder in the .zip file, to the directory of your choice (e.g. My Documents\MATLAB\OPCDA)
4. Open MATLAB and set this directory as the "current directory"



5. Open **OPCDARead.m**, which consists of a function that connects to a PI OPC DA Server on the specified machine, and read the current value of the specified tag(s).

   The essential parts of that code are:

```
% Initiate the connection
da = opcda('OPCServerNode', 'OSI.DA.1');
connect(da);

% Create a group the hold the item(s)
grp = addgroup(da);

% Add the item(s) to the group
additem(grp, '\\PIServerName\tagName);

% Extract the values from the OPC server
r = read(grp);

% Access the retrieved element(s): Value, ItemID, TimeStamp, Quality, Error
r.Value;      % or r(x).Value in case there were more than 1 item in the
group

% Disconnect
disconnect(da);
delete(da)
```

The OPC Toolbox documentation contains detailed examples about how to use the OPC DA functions it adds to MATLAB.

Note that the MATLAB OPC Toolbox only supports the OPC DA (Data Access) specification which is designed for **real-time data access**. Although this is designed for a quick and simple access to data for real time analysis, one could keep the connection with the OPC DA server alive for a long time and log that data - and hence achieve some sort of "historical data capabilities".

The OPC HDA (Historical Data Access) specification, on the other hand, allows retrieving and writing values in the past. To our knowledge, only one product exists for bringing OPC HDA connectivity into MATLAB: OPC HDA Toolbox by Integration Objects. Using that toolbox in conjunction with the PI OPC HDA Server theoretically represents a good solution to bring historical data from PI into MATLAB – *this was not tested for this paper*.

Using Integration Objects' OPC HDA Toolbox and the PI OPC HDA Server

The OPC HDA Toolbox from Integration Objects seems to be the preferred option to transfer historical data between OPC HDA compliant systems (such as PI with the PI OPC DA/HDA Server) and MATLAB.

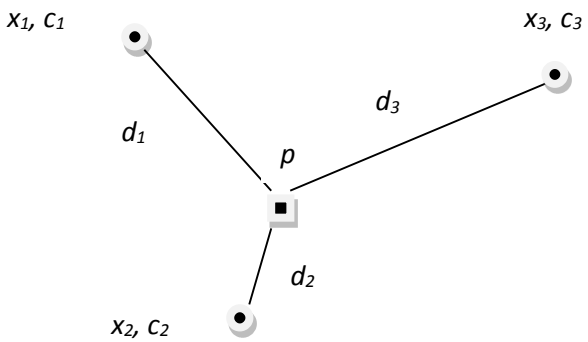
While this approach was not tested for this White Paper, we invite you to visit http://www.integ-objects.com/produits.php?id_produit=d888198688401b5cd37ec355953ec0fd for more information on the product.

Simulation: Optimizing Production Levels Using MATLAB Optimization Toolbox

In this section we model a production enterprise and optimize the production level in real time based on current costs and the selling price of the product. We use the programmatic way of connecting PI with MATLAB in Visual Basic .NET.

Assume there are three production plants producing x_i units of the same good each, where i=1,2,3 is the index of the site. Our goal here is to find the profit-maximizing production level at each site. The per-unit cost of production at site i is represented by c_i. Due to geographical and environmental factors, these costs could be time-varying and different from each other. The market, to which all the goods have to be shipped, is located at a location which has distance d_i from site i. As such the per-unit cost of transportation from location i to the market is td_i where t would typically depend on the fuel cost which varies over time. Given the above description, the cost of delivering one unit of the good to the market from site i is equal to $(c_i+td_i)x_i$.

The market price for one unit of the good is p. Therefore, the revenue would be equal to $p.(x_1+x_2+x_3)$. This gives the net profit as $\sum (p - c_i + td_i)\, x_i$.



In almost all such problems there are constraints on the production levels. Assuming we cannot consume the good at the sites, we should have $x_i \geq 0$. There is usually a maximum production level associated with each site $x_i \leq x_i^{max}$. To make the problem more interesting, let us assume there is another enterprise-wide limitation on the aggregate production level as well $x_1+x_2+x_3 \leq x^{max}$.

This brings us to this optimization program in three variables x_i, $i=1,2,3$:

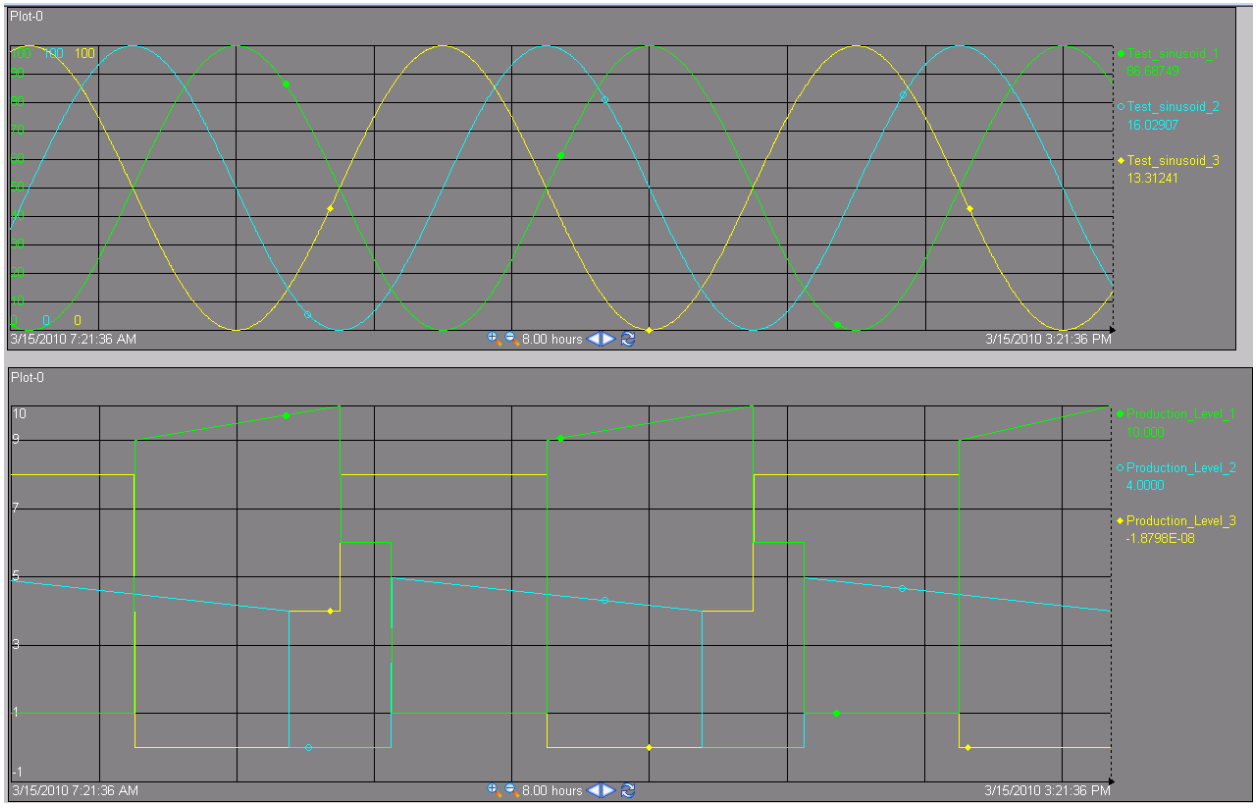
Max $\sum (p-c_i-td_i)x_i$

s.t. : $0 \leq x_i \leq x_i^{max}$

$x_1+x_2+x_3 \leq x^{max}$

This defines a simple linear optimization program in the production levels whose result would maximize the net profit. The time-varying parameters p, c_i, t, x_i^{max}, x^{max} could easily be stored in PI tags. The results, being the optimal production levels, could also be written back to some other PI tags. MATLAB on the other hand can very quickly and efficiently solve these kinds of optimization programs. In fact, using the same methodology, MATLAB and PI can represent, solve, and archive much more complicated optimization and scheduling problems on a regular basis. Hence, each site can react to market and environmental changes in real time resulting in more profit.

In an example, we assume that all production and business parameters are stored in PI and therefore we pull the per-unit profits of the three production sites from three PI tags and apply the optimization every 10 seconds using a timer in VB .NET. We also wrote back the resulting optimal production levels into three PI tags using PI-SDK. We used values 10, 5, and 8 for the maximum individual production levels and 14 for the sum of all production sites.



The top graph shows the per-unit profits of the three units while the lower trend shows the optimal production level in real time.

We let the program run for some time and plotted the simulation results. The bottom graph depicts the optimal production levels for the three sites at any instant of time as the per-unit profits fluctuate in the top graph. As the profit levels surpass each other the optimization program adjusts the production levels in real time while satisfying all the physical and business constraints. The linear program configuration as well as the code that was developed to make the graphs could be found in the appendix.

It has to be emphasized that this model is made deliberately simple to prove the concept while keeping this article convenient to read. Both PI and MATLAB are scalable and powerful products capable of handling problems much bigger and more sophisticated than the one described above.

A more interesting version of the same problem is when the price is a function of the production levels itself. This will define a *nonlinear* optimization program. Under typical circumstances of a market, such a problem can still be very efficiently solved routinely by combining MATLAB optimization toolbox and PI archiving capabilities.

Appendix A: Optimization Example Details and the Code

The matrices corresponding to the linear program are as the following:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 1 & 1 \end{pmatrix} \qquad b = \begin{pmatrix} 10 \\ 5 \\ 8 \\ 0 \\ 0 \\ 0 \\ 14 \end{pmatrix}$$

$$c = \begin{pmatrix} -profit_1 & -profit_2 & -profit_3 \end{pmatrix}$$

The Visual Basic code used to create the simulation results is the following:

```
'This is the general linear optimization function
'c is the objective function vector of coeffiecients while b is that of the
'constraints. A is the matrix corresponding to the linear constraints.

Private Function Linear_Program_With_Matlab(ByVal m As Integer, ByVal n As
Integer, ByVal c() As Double, ByVal A(,) As Double, ByVal b() As Double) As
Double()

        Dim MatLab As Object
        Dim Result As String
        Dim oMATLABOUT As Object
        Dim Bounded_A(0 To m - 1, 0 To n - 1), Bounded_c(0 To n - 1), _
                  Bounded_b(0 To m - 1) As Double
        Dim i, j As Integer
        Dim x(0 To n - 1) As Double  'This will be the unknown vector

        MatLab = CreateObject("Matlab.Application")


        'Filling in optimization vectors and matrix
        For i = 0 To m - 1
            Bounded_b(i) = b(i)
            For j = 0 To n - 1
                Bounded_A(i, j) = A(i, j)
            Next
        Next
        For i = 0 To n - 1
            Bounded_c(i) = c(i)
        Next
'Solving the program in MATLAB

        MatLab.putworkspacedata("c", "base", Bounded_c)
        MatLab.putworkspacedata("A", "base", Bounded_A)
```

```
        MatLab.putworkspacedata("b", "base", Bounded_b)

        Result = MatLab.execute("x=linprog(c,A,b)")

'Retrieving the result from MATLAB and returning the value

        Result = MatLab.getworkspacedata("x", "base", oMATLABOUT)
        For i = 0 To n - 1
            x(i) = oMATLABOUT(i, 0)
        Next i

        Linear_Program_With_Matlab = x

    End Function



'This is the function that executes at every ticking interval of the timer
'and calls the optimization procedure

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

'Setting up variables associated with PI tags

        Dim _pisdk As PISDK.PISDK = New PISDK.PISDK()
        Dim pt1 As PISDK.PIPoint =
_pisdk.Servers("afattahid630").PIPoints("test_sinusoid_1")
        Dim pt2 As PISDK.PIPoint =
_pisdk.Servers("afattahid630").PIPoints("test_sinusoid_2")
        Dim pt3 As PISDK.PIPoint =
_pisdk.Servers("afattahid630").PIPoints("test_sinusoid_3")

        Dim ptResult1 As PISDK.PIPoint =
_pisdk.Servers("afattahid630").PIPoints("Production_Level_1")
        Dim ptResult2 As PISDK.PIPoint =
_pisdk.Servers("afattahid630").PIPoints("Production_Level_2")
        Dim ptResult3 As PISDK.PIPoint =
_pisdk.Servers("afattahid630").PIPoints("Production_Level_3")


'Filling up linear program matrices

        Dim c(0 To 2), A(0 To 7, 0 To 2), b(0 To 7), x(0 To 2) As Double
        Dim m, n As Integer
        Dim c1, c2, c3 As Double


        c1 = pt1.Data.Snapshot.Value
        c2 = pt2.Data.Snapshot.Value
        c3 = pt3.Data.Snapshot.Value

        c(0) = -c1
        c(1) = -c2
        c(2) = -c3

        A(0, 0) = 1
        A(0, 1) = 0
        A(0, 2) = 0
        A(1, 0) = 0
        A(1, 1) = 1
        A(1, 2) = 0
        A(2, 0) = 0
```

```
        A(2, 1) = 0
        A(2, 2) = 1

        A(3, 0) = -1
        A(3, 1) = 0
        A(3, 2) = 0
        A(4, 0) = 0
        A(4, 1) = -1
        A(4, 2) = 0
        A(5, 0) = 0
        A(5, 1) = 0
        A(5, 2) = -1

        A(6, 0) = 1
        A(6, 1) = 1
        A(6, 2) = 1


        b(0) = 10
        b(1) = 5
        b(2) = 8
        b(3) = 0
        b(4) = 0
        b(5) = 0
        b(6) = 14

        m = UBound(b) + 1
        n = UBound(c) + 1

        x = Linear_Program_With_Matlab(m, n, c, A, b)

'Writing the results back to PI

        ptResult1.Data.UpdateValue(x(0), "*")
        ptResult2.Data.UpdateValue(x(1), "*")
        ptResult3.Data.UpdateValue(x(2), "*")
    End Sub
```

Revision History

09-Jul-09	Initial version by Steve Pilon and Yves Gauthier
07-Aug-09	Development topics added by Steve Pilon
25-Sep-09	Sections on OPC HDA and JDBC added by Steve Pilon
02-Apr-10	Optimization example added by Ahmad Fattahi
22-Aug-14	Sections on PI AF SDK and PI Web API added by Daphne Ng