Historian GetSamples Method Code

Retrieve historian samples in scripts.

When you create a trend chart and have data logging defined on **Historian / HistorianTags** or are using an AVEVA PI System, the product will automatically look for the archived data to plot the chart.

When you want to draw the trend chart mapping to external data, you have the following options:

- You can populate the Historian archive (SqLite Database, Microsoft SQL, PI System, or another database) from another application
- You can customize the GetSamples method used by the trend chart when plotting the data

Understanding How to use GetSamples Method
Customizing Getting Samples
Configuring the GetSamples Method
Example Using GetSample Method

Understanding How to use GetSamples Method

Customizing Getting Samples

The trend chart object calls the Historian server to get the data to plot the charts. In some scenarios, you may want to override that configuration and define a .NET code to provide the values. This is used, for instance, to plot recipe calculated data, future data, data from other SQL tables, or any custom scenario.

The custom GetSamples method should be defined for any Script Class, and this method must be defined for the column GetSamplesMethod on **Historian / HistorianTags**.

> The prototype of the method is:
> DataTable GetSamples(string[] tagNames, object startRange, object endRange)

When creating tables that are used on time charts, the StartRange and EndRange are DateTimeOffset type. When getting data to X-Y charts, the Range arguments are double variables.

The Returned DataTable for time Charts shall have the columns:

- DateTime: Date and time of the sample
- TagName: The name of the tag is used as the FieldName to the column with Double values of the tag.
- _TagName_Q: Optional column with the quality of the data

The Returned DataTable for XY Charts shall have the columns:

- X: The name X is used for the column with Double values
- TagName: The name of the tag is used as the FieldName to the column with Double values of the tag
- _TagName_Q: Optional column with the quality of the data

Configuring the GetSamples Method

The first step is to create the method itself. On **Scripts / Classes / ServerMain**, the following method must be created:

DataTable GetSamples

```
public DataTable GetSamples(string[] tags, DateTime start, DateTime end, TimeSpan interval, out bool
hasMoreSamples)
{
    // Initialize the DataTable that will be returned
    DataTable samplesTable = new DataTable();

    // Here you should add the logic to populate the DataTable
    // Example: samplesTable = GetSamples(tags, start, end, interval);

    // Assign a value to the 'hasMoreSamples' out parameter
    hasMoreSamples = false; // or true, depending on your logic

    // Return the DataTable
    return samplesTable;
}
```

Now, we need to fetch data from the external database. This can be done by executing a select command in a Query and saving the result on a temporary table.

In this example, we will fetch data from a table called ”externalDbValues" that is formatted as follows:

Name	Type	Max Length	System Type	Encoding	Null
Columns					
Code	Int	4	System.Int32		No
TempFurnace	Int	4	System.Int32		No
TempTank	Int	4	System.Int32		No
FanSpeed	Int	4	System.Int32		No
GenRotation	Int	4	System.Int32		No
Date	DateTime	8	System.DateTime		No

The SQL Statement executed in our class will be:

```
int sts = 0;
string msg = "";
@Dataset.Query.FetchData.SqlStatement  =  @"SELECT  *  FROM  externalDbValues";
DataTable TemporatyTable = @Dataset.Query.FetchData.SelectCommandWithStatus(out sts, out msg);
```

For the next step, we must start creating our final table with the same format as the default Historian Table. The columns that are required for the GetSamples method to work properly are:

1. Timestamp (in ticks) or DateTime
2. Item Value
3. Item

UTCTimestamp_Ticks	PLC_Value1	_PLC_Value1_Q	PLC_Value2	_PLC_Value2_Q
637057199434280000	0	0	0	0
637057199438410000	0	64	0	64
637057199468410000	0	64	0	64
637057199678980000	0	64	0	64
637057199709150000	0	64	0	64
637057199739160000	0	64	0	64

A similar table is manually created using the code below:

hasMoreSamples

```
hasMoreSamples = false;
DataTable table = new DataTable("Values"); table.Columns.Add("DateTime", typeof(DateTime));

foreach (string tag in tags)
{
        table.Columns.Add(tag, typeof(double)); table.Columns.Add("_"  +  tag  +  "_Q",  typeof(double));
}
```

Now, we need to populate the newly created table with the information fetched from our external database and return that table to our project.

```
foreach (DataRow foundRow in TemporatyTable.Rows)
{
        DataRow row = table.NewRow(); foreach (string tag in tags)
        {
                row[tag] = TConvert.To<double>(foundRow["<Column Name - Tag Value>"]);
                row["DateTime"] = new DateTime(TConvert.To<long>(foundRow["UTCTimestamp_Ticks"]));
                row["_" + tag + "_Q"] = 192;
        }
        table.Rows.Add(row);
}
return table;
```

At '*foundRow["<Column Name - Tag Value>"]* ', the inserted column name will be the one with the data that will be trended.

To plot a tag value into a TrendChart element, you will need to create a Tag, add it to Tag Historian Table, and insert it to the Pens list.

In order to avoid saving the fetch sampled data to the local historian database, you can disable the "Save On Change" configuration option.

You also need to add the newly created method into the GetSamplesMethod field.

Example Using GetSample Method

In the example below, the procedure to display DB data in a Trend is presented.

In Script / Classes, create a class and add the method below:

GetSamples code example

```
public DataTable GetSamples(string[] tags, double start, double end, double interval, out bool
hasMoreSamples)
{
        DataTable table = null;
        hasMoreSamples = false;

        int sts = 0;
        string msg = "";

        // Get DataTable
        DataTable TemporatyTable = @Dataset.Query.GetTable.SelectCommandWithStatus(out sts, out msg);

        table = new DataTable("Values");
        // This column name MUST be "X" to refer to X-Axis when the X-axis is an interval of the variable.
When the X axis is DateTime, is necessary to use "DateTime"
        table.Columns.Add("X", typeof(double));

        foreach (string tag in tags) {
                // Is necessary always the column to the name of the Tag
                table.Columns.Add(tag, typeof(double));
                table.Columns.Add("_" + tag + "_Q", typeof(double));
        }
        string nameMyTag = @Tag.MyValueY.GetName().Substring(4);

        foreach (DataRow foundRow in TemporatyTable.Rows) {
                DateTime date = new DateTime(TConvert.To<long>(DateTime.Now));
                DataRow row = table.NewRow();

                foreach (string tag in tags) {
                        @Info.Trace("Tag Name: " + nameMyTag);
                        row["MyValueX"] = TConvert.To<double>(foundRow["Value"]);
                        row["_" + tag + "_Q"] = 192;
                        row["X"] = TConvert.To<double>(foundRow["AxisX"]);
                }
                table.Rows.Add(row);
        }
        return table;
}
```

or

The code initializes a `DataTable`, retrieves a temporary `DataTable` using a query, creates a new `DataTable` with an "X" column, adds columns for each tag and associated data, populates the table with data from the temporary table, and returns the final `DataTable`.

GetSamples code example

```
public DataTable GetSamples(string[] tags, double start, double end, double interval, out bool
hasMoreSamples)
{
    // Initialize the DataTable
    DataTable table = new DataTable("Values");
    hasMoreSamples = false;

    int sts = 0;
    string mag = "";

    // Get the temporary DataTable from the query (assuming this query returns some results)
    DataTable temporatyTable = @Dataset.Query.GetTable.SelectCommandWithStatus(out sts, out mag);

    // Add the "X" column, which must be "X" to refer to the X-Axis when the X-axis is an interval of the
variable.
    // If the X axis is DateTime, it is necessary to use "DateTime" instead.
    table.Columns.Add("X", typeof(double));

    // Add columns for each tag
    foreach (string tag in tags)
    {
        table.Columns.Add(tag, typeof(double));
        table.Columns.Add(tag + "_Q", typeof(double));  // Column for additional quality/quantity data
    }

    // Iterate over the rows in the temporary table to populate the new table
    foreach (DataRow foundRow in temporatyTable.Rows)
    {
        DataRow row = table.NewRow();  // Create a new row for the table

        // Populate the row with data for each tag
        foreach (string tag in tags)
        {
            row[tag] = Convert.ToDouble(foundRow["Value"]);  // Assuming "Value" exists in the foundRow
            row[tag + "_Q"] = 192;  // Example fixed value, can be changed based on actual logic
        }

        // Populate the X column (assuming "AxisX" exists in the foundRow)
        row["X"] = Convert.ToDouble(foundRow["AxisX"]);

        table.Rows.Add(row);  // Add the row to the table
    }

    return table;
}
```

A Tag can be created to be used as a Pen within the Trend. To get values from a table or elsewhere, this tag needs to be added to the Historian (Historian / Historian Tags). After that, create the GetSamples(...) method within the ServerMain class. This method is where the values to be displayed on the chart will be placed. In this example, on line 27, the table value is added to the Tag, which is our Pen.

The values are added to the X-axis, which we also get from a column in the table. In this line, the row["X"] needs to be X so that WD can internally recognize that it is an XY graph.

Additionally, you need to go to Historian / Historian Tables, in the GetSampleMethod column and add the class and method that was created. This table is related to the tag that was created.

To add more tags within the same table, the tags need to be added in this way to the GetSample script:

```
//The code below processes each tag in the list, logs the tag, and populates a data row with corresponding
values from a source dataset.

foreach (string tag in tags)
{
        @Info.Trace("Tag Name: " + nameMyTag);
        row ["MyValueX"] = TConvert.To<double>(foundRow["Value"]);
        row ["MyValueY"] = TConvert.To<double>(foundRow["ValueY"]);
        row ["MyValueZ"] = TConvert.To<double>(foundRow["ValueZ"]);
        row ["_" + tag + "_Q"] = 192;
        row ["X"] = TConvert.To<double>(foundRow["AxisX"]);
}
```

In this section...
