Hyper Historian
Function Libraries

APPLICATION NOTE

August 2014

Description: Guide to creating your own function libraries in Hyper Historian.
General Requirement: Basic knowledge of .NET.

Introduction

Hyper Historian supports performance calculations based on ICONICS expressions infrastructure. The expression infrastructure has been mainly designed for real-time data processing while the performance equations are for working with historical data (collections of data values, etc.). Historical calculations are usually very complex and are often not easily human-readable expressions. As an easier approach to performance calculation support, the concept of external .NET based libraries has been developed.

This application note intends to get the user started with function libraries. Basic knowledge of programming in .NET is expected. For more information refer to the help documentation or contact ICONICS Technical Support.

Design

A function library is represented by a public .NET class where all public methods can be used as "calculation methods" inside Hyper Historian. A single assembly can contain one or more function libraries (public .NET classes).

Creating Function Libraries

Library functions have to follow simple rules given by the Hyper Historian infrastructure:

- Functions have to be declared as **public.**
- Functions have to have **unique names** within library class.
- Functions can be declared as static.
- Functions **have to return a value** – this value is then either logged or used by other functions.
- Functions **can be invoked by multiple threads at the same time**. Do not lock functions – it may decrease overall calculation performance. Instead consider the use of separate contexts for each function call, if needed.

Functions without Internal Access to Historical Data

Some functions can be created without internal access to historical data. This approach assumes that all necessary historical data is passed to this function as parameters, thus no direct historical access is required. These functions can be defined as static functions. The following example code shows the simplest function for addition of two double values:

```
namespace HHfunction
{
        public class MathLibrary
        {
            public static double Addition(double a, double b)
            {
                return a + b;
            }
        }
    }
```

Figure 1 - Simple summary function

All necessary .NET types should be available in the following assemblies:

- HHPlugIn.dll
- HHCommon.dll

Though very simple libraries may not need to refer to any Hyper Historian specific assemblies, typical function libraries will require referencing both of the above mentioned assemblies. Simply go into Project settings and select Add Reference – here you can add/remove or check all available .NET types.

If you need to know some context information such as the execution time, trigger name or previous value add a context parameter to the library function. This parameter must be specified as the first parameter and has to be type of IEvaluatorContext:

```
public static double Addition(IEvaluatorContext context, double a, double b)
{
    return a + b;
}
```

Figure 2 - Example context parameter

If the user wants to get a complete logged value for further processing then the given parameter has to be the type of LoggerDataValue.

```
public static double Addition(IEvaluatorContext context, LoggerDataValue a, LoggerDataValue b)
{
    if (a.ValueStatus.Good && b.ValueStatus.Good)
        return LoggerVariant.ToDouble(a.Value) + LoggerVariant.ToDouble(b.Value);

    throw new Exception("One of the input value is BAD");
}
```

Figure 3 - Example LoggerDataValue parameter

Similarly, to get an array of logged values specify parameter as IList<LoggerDataValue>.

Copyright 2014 ICONICS, Inc.

```
public double MyMethod(IEvaluatorContext context, IList<LoggerDataValue> value)
{
    // do some calculations
    return 0;
}
```

Figure 4 - Example IList<LoggerDataValue> parameter

Function with Internal Access to Historical Data

These functions have internal historical access similar to standard "History Functions" defined in Hyper Historian for the expression engine. Their parameters should only contain data point identifications, and the function should take care of data reading and processing. The "FunctionLibrary" base class has been developed to simplify the historical access.

```
public class MathLibrary : FunctionLibrary, IDisposable
{
    public MathLibrary(ILibraryDataAccess dataAccess)
        : base(dataAccess)
    {
    }

    // add functions here...

    public void Dispose()
    {
        // do cleanup
    }
}
```

Figure 5 – Creating a new FunctionLibrary

The FunctionLibrary base class implements miscellaneous methods for reading data from the historical storage. Data points are identified in all methods by a parameter of type IDataVariableNode. As an example, a method processing time weighted average of logged data samples may look like the following:

```
public LoggerDataValue TimeAverage(DateTime startTime, DateTime endTime, IDataVariableNode point)
{
    return ReadResampledValue(
        point,
        ResampledValueType.TimeAverage,
        startTime, endTime,
        80,
        false);
}
```

Figure 6 - Time-weighted average example

Sometimes it is advantageous to dynamically construct a point object of type IDataVariableNode, perhaps based on a string or numeric parameter. Then in the function we would need to try to get or construct the object (IDataVariableNode) before reading its data. To accomplish this, the following method is available, where "XXX" presents the Hyper Historian data point:

```
if (DataAccess.TryGetDataVariableNode("data:XXX", out point))
{
    // do some point reading
}
```

Figure 7 - Get a data variable node

Registration of Function Libraries in the Hyper Historian configuration

The new function library needs to be built in Visual Studio, then the created .dll file needs to be copied to the ICONICS Component folder.

To enable a particular method for use in Hyper Historian a new Calculation Function has to be added in the Hyper Historian configuration in the "Calculation Function" section. The parameters of the calculation function should be specified. The number of parameters has to match the number of arguments declared in the .NET method with one exception – if the very first parameter is of type IEvaluatorContext it will be set by the Hyper Historian infrastructure and should not be defined in the custom function.

Once all parameters are configured, the Expression tab of the Calculation Function should specify the custom function using the following syntax:

{{lib:*Assembly_name*!*Class_name*:*Function_name*}}

Where:

- ***Assembly_name*** is the actual assembly name (example: MyLibrary.dll). If the name doesn't contain directory info then Hyper Historian assumes this library is stored in ICONICS Component folder.
- ***Class_name*** is a class name including namespace (example: MyLibrary.MyClass). The class must be public.
- ***Function_name*** is the actual function name inside specified class (example: MyFunction).

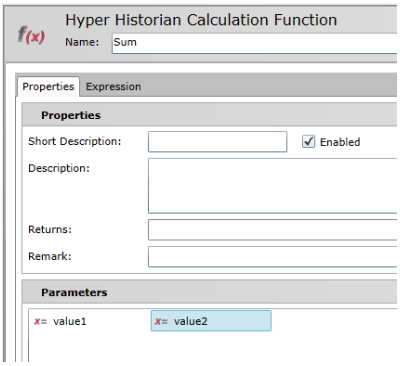

Figure 8 - New calculation function with parameters

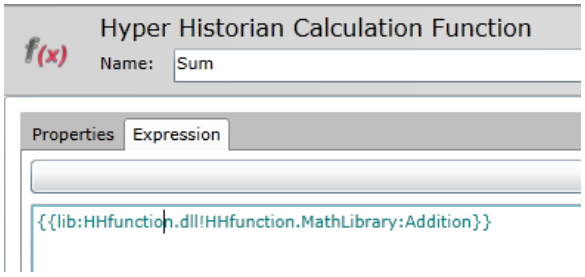

Figure 9 - Special expression text

Using Function Libraries for Logging

All calculation functions created according to the steps above can be reused for value logging.

Simply create new Calculated Tag or Calculation Function and browse for the newly created Calculation Function under Functions → Calculation Functions.

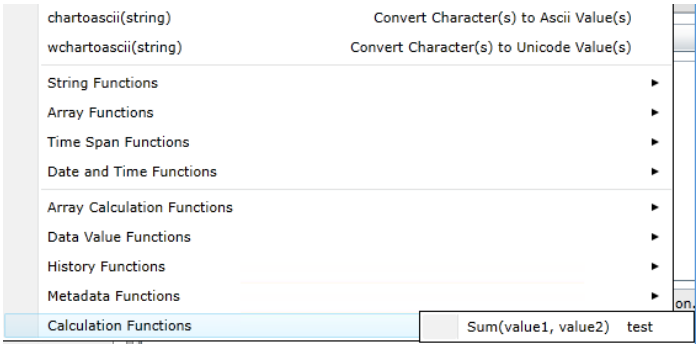

Figure 10 - Browsing calculation functions

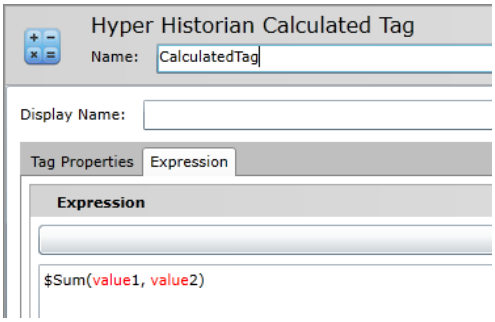

Figure 11 - Parameters definition

Debugging

The easiest way to debug any custom library is to use Visual Studio 2010 or higher and attach it to the logger process. To simplify development, a developer can select the function library as the startup project and in the project's properties → debugging select "external program" and browse for HHLoggerHost.exe (which is located by default in the Components folder). Make sure to copy the function library into the Component folder (or the folder which contains the Hyper Historian assemblies). This can be done in post-build steps.

Deployment

Function library should be deployed as set of required assemblies plus header files, exported from the Hyper Historian configuration. The header (descriptor) files can be exported from the Hyper Historian configuration in Workbench. Go to Calculation Functions, select a Calculation Folder, right click it and select Export Library.

To deploy it on a different PC, first copy all the assemblies to the ICONICS Components folder and then import the library header files. Once the import is completed all library functions should be available for use in calculated tags.