EcoStruxure™

Geo SCADA Expert

Data Feeder

Sample Data Export System

www.schneider-electric.com

****

SAMPLE CODE FOR EDUCATION ONLY. THIS CODE MAY NOT BE CYBER-SECURE.
YOU ARE RESPONSIBLE FOR IMPLEMENTING CYBER-SECURITY FEATURES.

****

Issue Details

Issue	Date	Author	Comments
1	06.Feb.23	S. Beadle	Update security warning
2	23.Apr.23	S. Beadle	Add Sparkplug Data Feeder with enhanced features
3	08.June.23	S. Beadle	The Sparkplug Data Feeder now runs as a Windows Service

References

Geo SCADA.Net Client:

Help file installed with Geo SCADA and available in the Start Menu as “Client API Guide”.

OSISoft Message Formats: https://docs.osisoft.com/bundle/omf/page/index.html

MIT License

Table of Contents

1 Introduction

This document describes the code for a set of sample data export programs for Geo SCADA Expert, written in C# for the .Net Client API.

The data feeder software is offered as source code which you can build with Visual Studio. It includes a core 'FeederEngine' library and sample data feeder programs, all can be modified by you. It is not supported by Schneider Electric.

The source code is available for you to freely use, modify and extend to suit your requirements or that of your clients. It is not the most optimized, efficient or elegant code, and the functionality is not assured in the way the core product is, but we hope that its simplicity will encourage engagement with the Geo SCADA extension development process and explore the new ideas presented. We encourage you to add to the code by submitting 'pull requests' on GitHub.

The purpose of the Data Feeder is to give a reasonably efficient way of exporting point data (value, quality and timestamps) to another system external to Geo SCADA. The export can be configured, so you select which points are to be exported and the characteristics of the data to be exported.

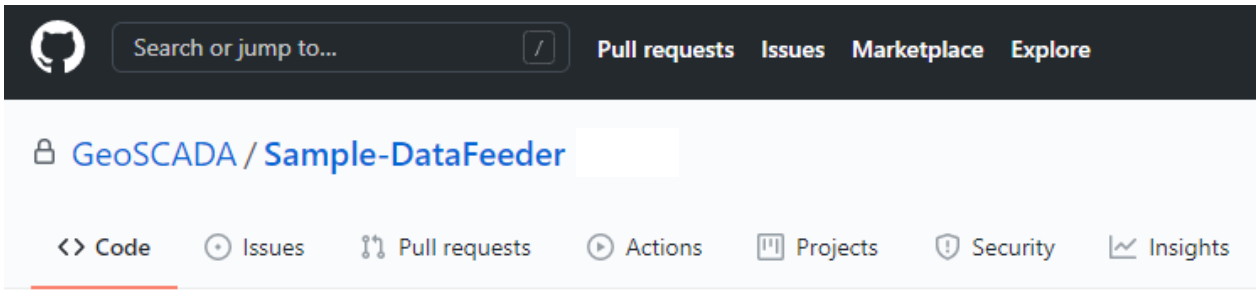
To implement and deploy this example you will need to verify functionality and add appropriate security measures for your environment.

You can discuss these features in the SE Exchange forums:

https://community.exchange.se.com/t5/Geo-SCADA-Expert-Forum/bd-p/ecostruxure-geo-scada-expert-forum

1.1 The Source Code

You will find source code for this driver within the GitHub system. The project name is Geo SCADA / Sample-DataFeeder:



URL: https://github.com/GeoSCADA/Sample-DataFeeder

1.2 The Projects

There are multiple projects included:

a) A FeederEngine project which builds a .dll for use by a project. Each project includes this engine.

b) A file-based export project which uses FeederEngine to write text files of data updates containing JSON data

c) A Microsoft Azure Event Hub export project which uses FeederEngine to send JSON data to Azure

d) An MQTT export project which uses FeederEngine to send JSON data to Azure

e) An OSI Pi and OSISoft Cloud Services (OCS) project which uses FeederEngine to send JSON data in OSIsoft Message Format (OMF) on-premises Pi and OCS systems.

f) A Sparkplug Client Feeder which includes a Service, setup file, redundancy and many other features not found in the other samples. See the Sparkplug Feeder section of this document for more information.

You will need the FeederEngine (a) and your choice of export project, which could start as the other projects and be modified to your requirements. For example, to export to MQTT, other cloud systems or historians.

1.2.1 Data Feeder uses the .Net Client

The feeder uses the Advanced section of the client, which adds 'on change' functionality to points. In other words, if a point does not change then the data export process does not waste time interrogating the database and writing out unnecessary data.

The feeder also uses the Geo SCADA historian to export all changed data since the last export, ensuring 'gap fill' of data. It will do this for every change period, so there is no need for the change period to be configured as a short interval, which will compromise performance.

When adding a new point to the feeder (all points to be exported need to be added), it is possible to specify the earliest time for which historic data will be interrogated and exported. While the examples use a default time, your implementation could read a 'last exported' time from local storage or some other system in order to ensure that all gaps are filled. Writing of the last data time into this store is not included in the samples.

The feeder can run on any client and connect to any server. For performance reasons you may with the feeder to:

a) Connect to a Standby or Standby-Only server to reduce load on a Main server.

b) Run locally on that server, instead of a separate client, so that Geo SCADA communications are within-server.

The example code does not provide any redundancy for multiple connections, but it would be simple to add if required. There is a server failure/shutdown handler.

1.2.2 Message Content

The code defines the fields used for two types of message. The first is a Configuration message, which just contains point full name and Id, but could be expanded to include other information. This is sent when a point is added to the engine (optional) and is also added when a new point is added to Geo SCADA which meets the requirements of export.

The second type is a data update message. This could contain historic data value, quality and time, or if the point is not stored historically, then a current value at the time of change.

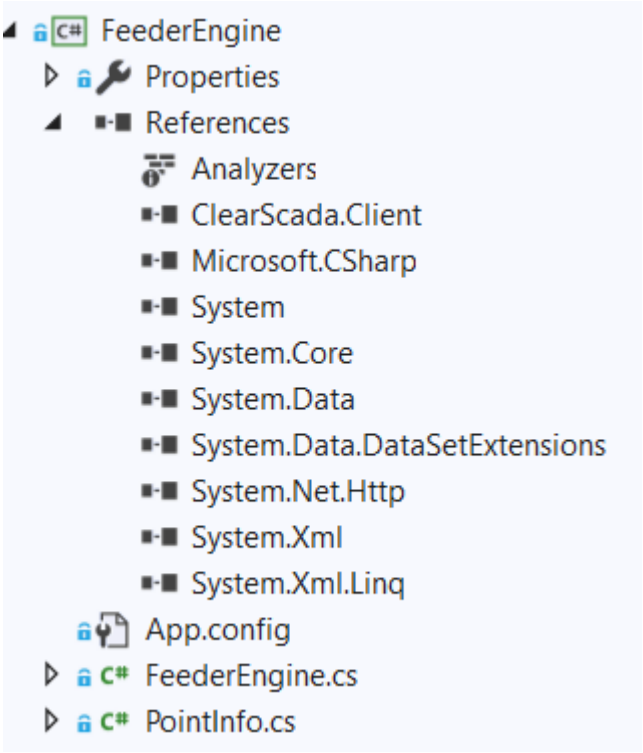
2 Building the Code

This section lists the projects within the solution and notes on how to build them.

2.1 FeederEngine

This is the .dll module which defines how the export works. You can still change this code as required.

2.1.1 References



ClearSCADA.Client

The module refers to the Geo SCADA dll 'ClearSCADA.Client.dll'. You should find this reference in the project, remove it and then reinstate it from the location used on your build computer.

If you are developing on a computer which has Geo SCADA installed, find this in c:\Program Files\Schneider Electric\ClearSCADA

Alternatively, you can copy this .dll to your own location from a Geo SCADA installation. In this case you may set up multiple builds for different Geo SCADA versions if you wish. Note that you need the version number of this .dll to match the major version of the Geo SCADA target.

2.1.2 Files

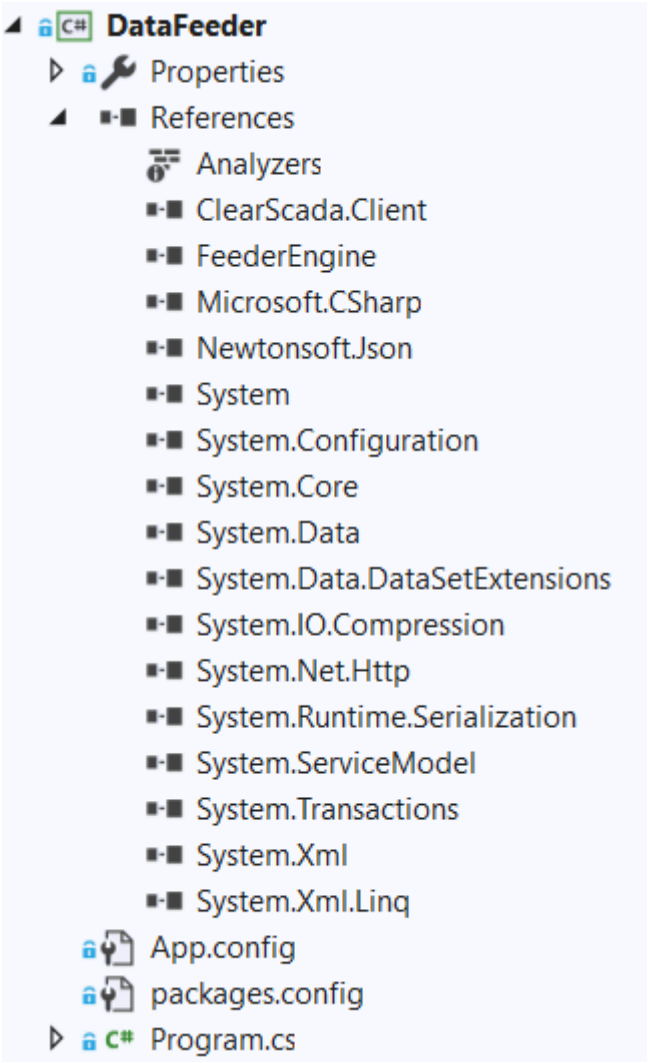
The file FeederEngine.cs is a static class defining the functions of the export process. Only one database connection can be used, and only called in a single thread.

The file PointInfo.cs contains a class used by the FeederEngine to retain information about each point (or accumulator) to be exported.

2.2 DataFeeder

This is the .exe file to be customized to export data to your choice of target and format. If you wish, you could create a version which works as a system service, or even include its functionality in a DDK Driver.

2.2.1 References



ClearSCADA.Client

The module refers to the Geo SCADA dll 'ClearSCADA.Client.dll'. You should find this reference in the project, remove it and then reinstate it from the location used on your build computer.

If you are developing on a computer which has Geo SCADA installed, find this in c:\Program Files\Schneider Electric\ClearSCADA

Alternatively, you can copy this .dll to your own location from a Geo SCADA installation. In this case you may set up multiple builds for different Geo SCADA versions if you wish. Note that you need the version number of this .dll to match the major version of the Geo SCADA target.

3 Using the Data Feeder

This section describes how to get started with the driver.

Make changes to the file Program.cs in the project DataFeeder.

3.1 Set the Update Intervals

```
30          private static int UpdateIntervalSec = 300;
```

The functionality in the FeederEngine includes the 'registering' of an interest in each source point or accumulator, and callback functions when new data is available. The interval 'Current Update Rate' over which new data is read is a very important aspect of this.

This is the change detection interval for points. You are not recommended to speed this up much, as performance is impacted if this time is short. If you are feeding mostly points with historic data, then the PointInfo class will fill in the gaps with data queries and you can raise this detection interval for better performance.

In other words, if you can wait up to 15 minutes or more to receive and push data, then you should. Set this as high as you can. However, if you are feeding points with current data (i.e. their History checkbox is not enabled), then this interval is the minimum time interval that changes will be detected, and changes faster than the interval will be missed.

If you are scanning data sources every 30 seconds with NO history and you want every update, then set this interval to match, but be aware that system performance could be impaired. Do not use an interval of less than 30 seconds, that is not sensible. If you have a mix of historic and non-historic points, then consider modifying the library to read from each type of point at different intervals.

3.2 Set an Output Location for the Exported Files

```
37          // Test data file for output of historic, current or configuration data
38          private static string FileBaseName = @"Feeder\ExportData.txt";
```

This could be an absolute or relative reference. This location is used as the 'progress' CSV file which is saved each minute to enable the feeder to 'catch-up' on a restart and fill gaps.

The Sparkplug feeder is set up via a JSON file in ProgramData.

3.3 Add User Credentials

```
60          foreach (var c in "MyPassword1")
61          {
62              spassword.AppendChar(c);
63          }
64          AdvConnection.LogOn("FeederUserName", spassword);
```

These should be a user with just read access to Geo SCADA data.

You are recommended to encrypt and store these credentials outside the compiled code.

The Sparkplug feeder stores credentials for Geo SCADA and the MQTT Broker using files in ProgramData. You first add plaintext credentials in the JSON configuration file.

These are then encrypted on first-run into files in the ProgramData folder. You may then remove the credentials from the JSON file, and the encrypted credentials will be used from this time. The files are encrypted so that they cannot be used on another PC.

3.4 Refer to Points you wish to Export

```
80          var MyGroup = AdvConnection.FindObject("SA"); // This group id could be used to monitor a subgroup of points
81          await AddAllPointsInGroup(MyGroup.Id, AdvConnection);
```

This sample picks a group and monitors all points in that group by adding them recursively. The function to add them is Feeder.AddSubscription(Name, StartTime).

The functions AddAllPointsInGroup and AddPoints do this.

3.5 Add a Filter for New Points

The export mechanism waits for point configuration change and adds new points, which means that the export process does not need to be restarted when the database configuration is modified. A function FilterNewPoints is used as a callback to choose points meeting your set criteria.

3.6 Choose Export File Size

```
192         public static void ExportStream_WriteLine(string Out)
193         {
194             // Create a new file after <4K
195             if (ExportStreamBytesWritten > 4000)
196             {
197                 CloseOpenExportFileStream();
198             }
```

This sample writes files of export data in JSON format, using a simple size constraint to cause new files to be started.

This does not apply to the Sparkplug export.

3.7 Check the Export

When you run the program, the database is read and all points chosen to be monitored by the feeder will be added. If the second argument to the Feeder.Connect function is true, then all added points will have a ‘ConfigChange’ JSON record exported, with the UpdateType string set to “Added”.

```
67          // Set up connection, read rate and the callback function/action for data processing
68          if (!Feeder.Connect(AdvConnection, true, UpdateIntervalSec, ProcessNewData, ProcessNewConfig, EngineShutdown, FilterNewPoint))
69          {
```


```
{"data":[
{"PointId":116194,"UpdateType":"Added","PointName":"SA.Other.SameGame.Stuff.HS","Timestamp":"2021-07-07T09:09:18.0006761+00:00"}
,{"PointId":111874,"UpdateType":"Added","PointName":"SA.Plant.Energy.Backup Systems.System A.Digital","Timestamp":"2021-07-07T09:09:18.1578498+00:00"}
,{"PointId":111877,"UpdateType":"Added","PointName":"SA.Plant.Energy.Backup Systems.System A.Runtime","Timestamp":"2021-07-07T09:09:18.1588258+00:00"}
...
```

After the update interval (default 5 minutes) the export will output JSON records containing value, quality and time:


```
{"data":[
,{"PointId":112413,"UpdateType":"His","PointName":"SA.Plant.System
C.Measures.T1.Analog","Value":59.103518784142658,"Timestamp":"2021-07-
07T10:13:16.3477794+01:00","OPCQuality":192,"ExtendedQuality":1024}
,{"PointId":112413,"UpdateType":"His","PointName":"SA.Plant.System
C.Measures.T1.Analog","Value":58.556321909238029,"Timestamp":"2021-07-
07T10:13:18.3374412+01:00","OPCQuality":192,"ExtendedQuality":1024}
,{"PointId":112413,"UpdateType":"His","PointName":"SA.Plant.System
C.Measures.T1.Analog","Value":59.373149815363092,"Timestamp":"2021-07-
07T10:13:20.3396503+01:00","OPCQuality":192,"ExtendedQuality":1024}
,{"PointId":112463,"UpdateType":"His","PointName":"SA.Plant.System
C.Measures.T2.Analog","Value":8.1847590563676853,"Timestamp":"2021-07-
07T10:12:52.3436053+01:00","OPCQuality":192,"ExtendedQuality":2098176}
,{"PointId":112463,"UpdateType":"His","PointName":"SA.Plant.System
C.Measures.T2.Analog","Value":8.0504776146732979,"Timestamp":"2021-07-
07T10:12:55.4102425+01:00","OPCQuality":192,"ExtendedQuality":2098176}
...
```


The export process monitors queue size and checks whether the queue size is getting larger. Under normal operation the queue would be emptied in a timely way before new entries are added:

```
Total Updates: 0 Rate: 0 /sec Process Time: 0mS, Queued: 0
Total Updates: 0 Rate: 0 /sec Process Time: 0mS, Queued: 0
Total Updates: 983 Rate: 35.0317452829042 /sec Process Time: 1011.7725mS, Queued: 77
Total Updates: 1504 Rate: 49.9428721947887 /sec Process Time: 1053.1081mS, Queued: 59
Total Updates: 2560 Rate: 79.7074050757352 /sec Process Time: 1001.422mS, Queued: 22
Total Updates: 3171 Rate: 94.3935907615138 /sec Process Time: 473.52mS, Queued: 0
Total Updates: 3171 Rate: 91.6598863113322 /sec Process Time: 0mS, Queued: 0
```

4 Sparkplug Feeder

The Sparkplug feeder project has the following features in the code. Here is a summary, with more detail to be added to the document later:

- Windows Service setup. You may run as a normal program, or install and start as a Windows Service. See below.
- Settings file in a JSON format. Run the program the first time to save the settings file, then edit the settings file for future runs. The location defined in the code is: `c:\ProgramData\Schneider Electric\SpDataFeeder\Settings.json`
- Redundant connection to two Geo SCADA Servers – defined in the settings file
- Redundant connection to two MQTT Broker Servers – defined in the settings file
- Encrypted Geo SCADA and MQTT Broken user credentials in files. Run the program with command line parameters:

  "Geo-SCADA-Username" "Geo-SCADA-Password" "MQTT-Username" "MQTT-Password"

  Run once to write the usernames and passwords into the files then clear the parameters for future runs and the files will be used.
- Failure handling to pause/continue automatically if Geo SCADA or the MQTT broker fails. (Further try/catch handling may be needed for a complete solution).
- Edit the scope of exported data in the settings file. Use settings to control the group levels in the Sparkplug metric names.
- The export associates' data with a single Node, no Devices are used.
- Use a Logic program in Geo SCADA to cause your Node to automatically create points in your 'receiving' Sparkplug Node. See the sample .sde file in the Sparkplug project folder.
- The feeder will export configuration fields – use Geo SCADA's automatic configuration facilities to set these in the upper tier system. A sample Property Translation Table text file is included in the Sparkplug project folder.
- Logging to file and console using NLog. Find the logs in the folder: `c:\ProgramData\Schneider Electric\SpDataFeeder\Log\SpDataFeeder.log`

Known issues

- When a Birth certificate is processed without initial data values (which is the approach taken by this code), Geo SCADA will add a zero value to the point. This issue will be resolved in a future Geo SCADA update. A workaround is added which uses an old time-stamp for the value.

4.1 Windows Service

This is a Windows Service program. You can run/test it without installing as a service.

To install the service, run a CMD window as Admin, go to the compiled output folder and enter:
%windir%\Microsoft.Net\Framework64\v4.0.30319\installutil DataFeederSparkplug.exe

Note: if you have run the program without installing as a service, installation may be blocked because the event log Source already exists. First go to the Event Viewer and right-click DataFeederLog in apps & services logs, then clear it.

The service optionally takes a command parameter - the filename of the setup file.

You could use WIX to create an installer msi which completes the installation.

5 The Code

The code for the feeder module and sample program is written with onward development in mind. There are many comments, and a structure which you can take on and modify or extend.

5.1 Where Next?

Customize Text Export Format

The present format is a particular JSON schema – simple to change in the classes DataVQT and ConfigChange.

Customize Export Target

Samples are given for files and Azure Event Hubs. Customization of CreateExportFileStream or SendToAzureEventHub is needed.

Write other Configuration Properties

Extend the configuration export to write other point properties such as analog Units and ranges. Add to the function `ProcessNewConfig`

6 Appendix A – Setting Up Azure EventHub

This is provided in the form of screenshots showing the key steps: