



**EcoStruxure™**

**Geo SCADA Expert**

**Data Feeder**

**Sample Data Export System**

[www.schneider-electric.com](http://www.schneider-electric.com)

---

## Issue Details

Issue	Date	Author	Comments
1	06.07.21	S. Beadle	New.

### References

Geo SCADA.Net Client:

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

### MIT License

Copyright (c) 2021 Schneider Electric and its subsidiaries. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# Table of Contents

Chap	Title	Page
<b>1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	The Source Code .....	1
1.2	The Projects .....	1
1.2.1	Data Feeder uses the .Net Client.....	2
1.2.2	Message Content.....	2
<b>2</b>	<b>Building the Code.....</b>	<b>3</b>
2.1	FeederEngine .....	3
2.1.1	References.....	3
2.1.2	Files .....	3
2.2	DataFeeder .....	4
2.2.1	References.....	4
<b>3</b>	<b>Using the Data Feeder.....</b>	<b>5</b>
3.1	Set the Update Interval .....	5
3.2	Set an Output Location for the Exported Files .....	5
3.3	Add User Credentials .....	5
3.4	Refer to Points you wish to Export .....	5
3.5	Add a Filter for New Points .....	6
3.6	Choose Export File Size .....	6
3.7	Check the Export .....	6
<b>4</b>	<b>The Code.....</b>	<b>8</b>
4.1	Where Next?.....	8

# 1 Introduction

This document describes the code for a sample data export program 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 two 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.

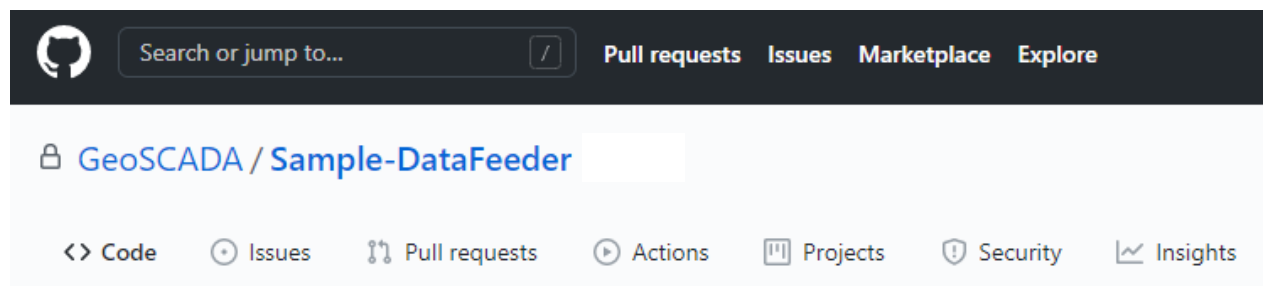
To implement and deploy this example you will need to verify functionality and add appropriate security measures for your environment.

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 three projects included:

- a) A FeederEngine project which builds a .dll for use by a project
- 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

You will need the FeederEngine (a) and your choice of export project, which could start as (b) or (c) 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 wish 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.

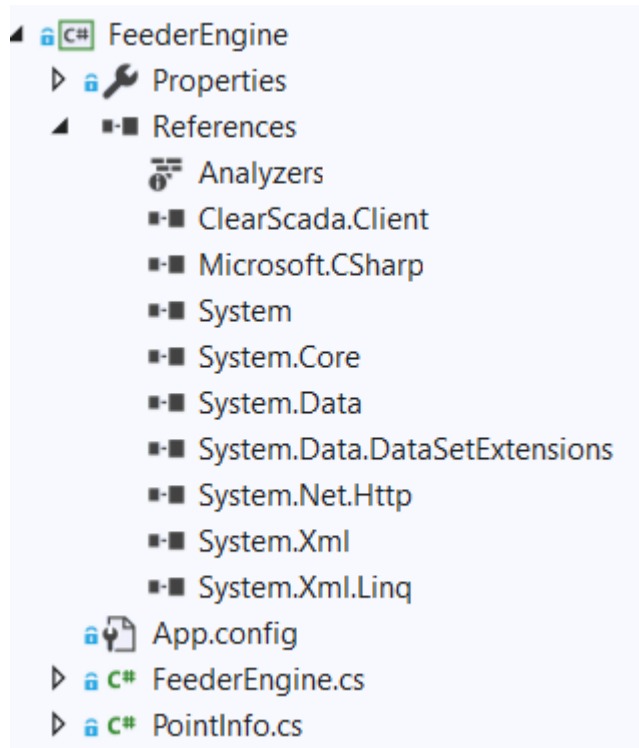
## 2 Building the Code

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

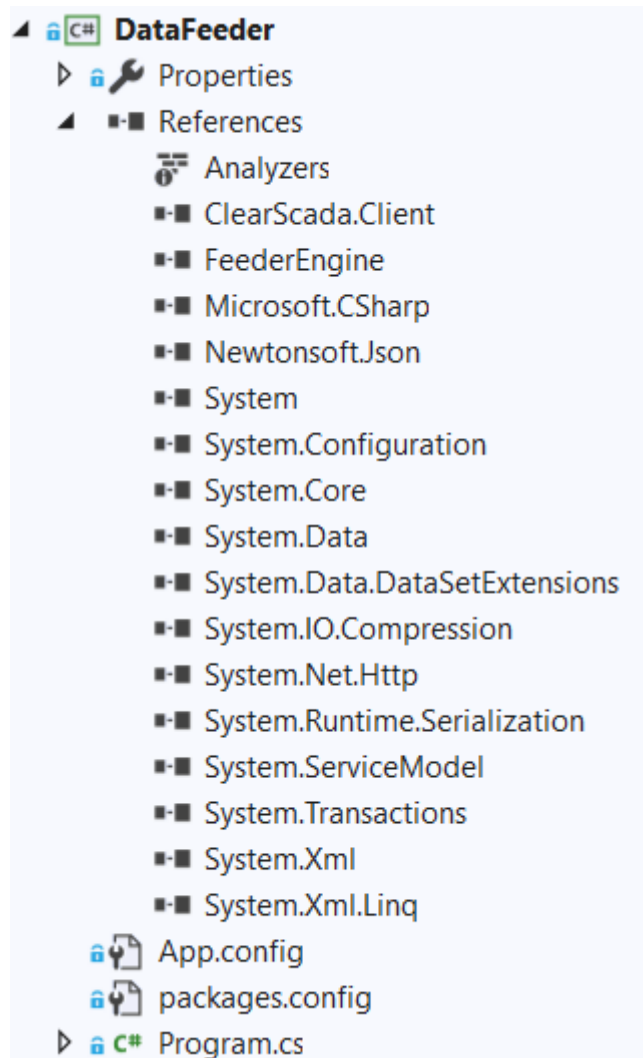
The file FeederEngine.cs is a static class defining the functions of the export process. Only one database connection can be used, and only called in a single thread.

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 Interval

```
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 FileName = @"Feeder\ExportData.txt";
```

This could be an absolute or relative reference.

### 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.

### 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. In this example it returns true, so all new points are added to the export process, even if they are not covered by the first call to the function `AddAllPointsInGroup`.

### 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.

### 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","Time
stamp":"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 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.

### 4.1 Where Next?

#### ***Customize 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