

*NSBS System for Storing Coil Data*

This system provides the means to store and retrieve coil data in a SQL2012 (or better) database.

Dan Houck (AS&E)

18 Oct 2014

Last Update 03 Nov 2015

CoilStore

NSBS System for Storing Coil Data

Contents

[Overview 2](#_Toc434315777)

[Operation 3](#_Toc434315778)

[The Program 4](#_Toc434315779)

[Appendix – Typical Folder Structure 6](#_Toc434315780)

[Appendix – Database Structure 7](#_Toc434315781)

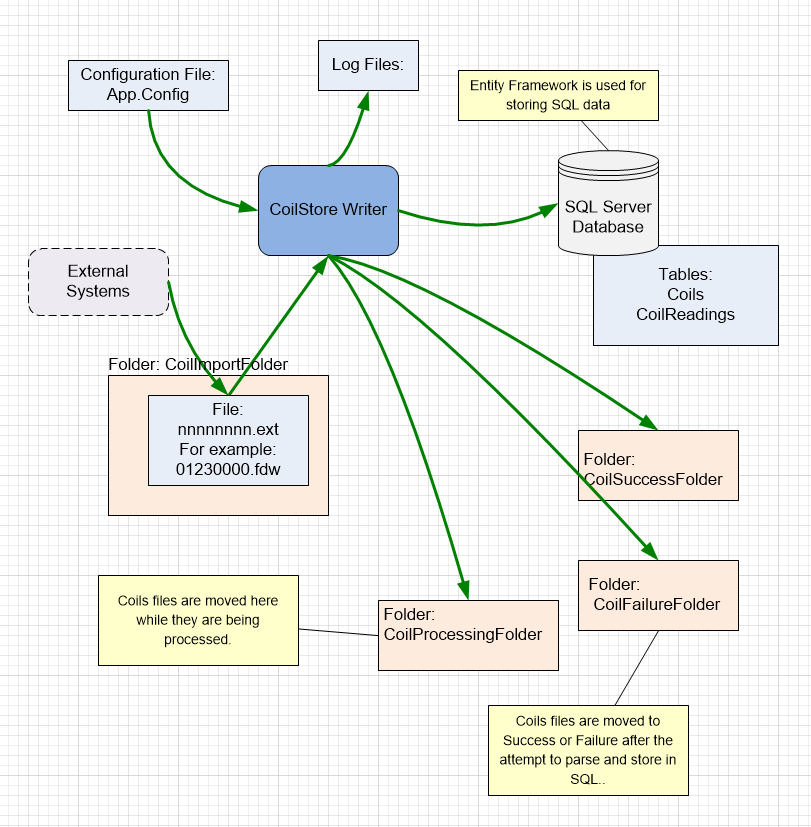
[Appendix - Raw Coil File 10](#_Toc434315782)

[Appendix – Using Entity Framework and EDMX in Visual Studio 11](#_Toc434315783)

# Overview

CoilStore is designed to take raw coil data reading files, parse the information, and place the data into a SQL Server relational database.

The following diagram illustrates the major points, which are discussed in more detail below.



Overview Diagram

# Operation

External systems place data for a coil as ASCII files into the coilImport folder. These files are expected to have a file name as an 8 digit number, and have coil collection types identified by their three character extension of either “fdh” or “fdw”. Examples are 01350000.fdh and 01350000.fdw.

More information on these files is available in the appendix.

The CoilStore process watches for files in this folder, and moves them to the file named by the setting CoilProcessingFolder (for example d:\(data)\CoilImport\Processing) so the files can be processed by the CoilStoreEngine. Upon processing the file is placed into one of the two subfolders:

|  |  |
| --- | --- |
| **Setting** | **Example** |
| CoilFailureFolder | d:\(data\CoilImport\Failure |
| CoilSuccessFolder | d:\(data\CoilImport\Success |

Files are watched using the system FileWatcher, but for insurance there is also a setting FilePollSeconds (example value: 15) that will look for the files periodically.

Any logging – informational or otherwise – is placed in a date-stamped file according to the setting LogFolder. An example value is d:\logs.

The data is parse and – if successful – placed in the SQL database. If a duplicate coil is found, then the existing coil is deleted and the new data replaces it.

The data is inserted with a constructed Coil Key (BigInt) that has the structure:

Yymmddhhmmssd,

Where the time is the time the coil was processed (received in the ASCII file), and the “d” on the end is 1 if the file extension was “fdh” and 2 if it was “fdw”.

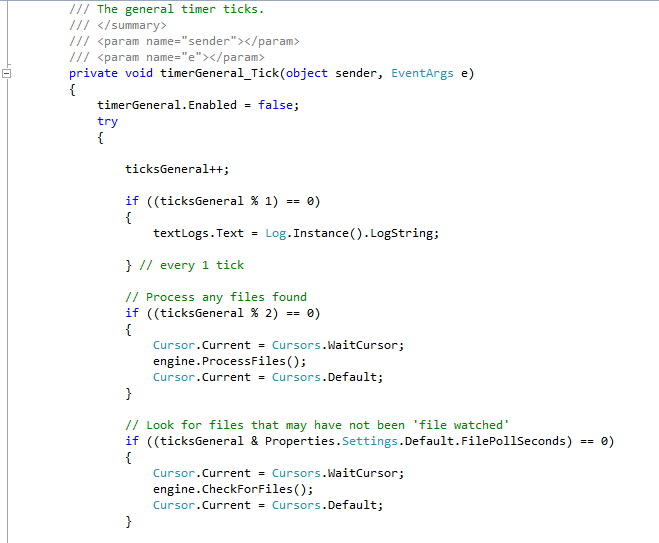
**Update (Nov 2015): The time received in the file is local time. The Nov 2015 update now converts this to UTC to prevent duplicates during the fall resumption of “standard” local time.**

All of the data is placed into the SQL Server database into two tables Coils or CoilReading. More information on these tables is available in the appendix.

## The Program

The program is a WinForms application. It sets up a FileWatcher to look for files being received, and also has a timer to check for files the FileWatcher might miss, and also checks the processing queue according to file FilePollSeconds setting.

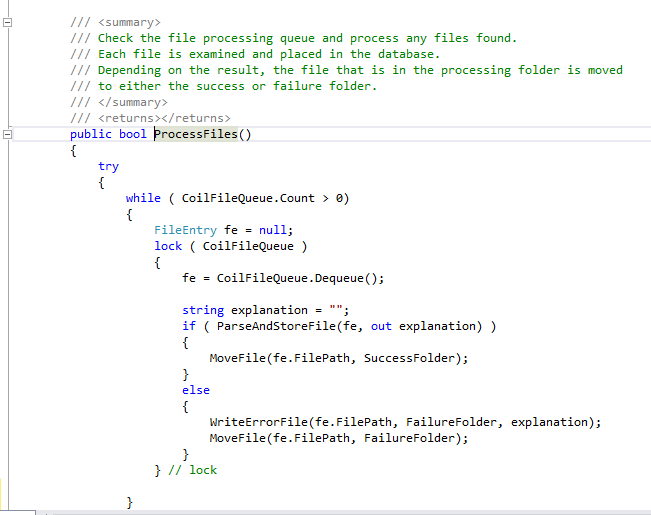
Here is a snippet for the timer method:



The important processing logic is contained within the CoilStoreEngine class, and the two most important methods are:

|  |  |
| --- | --- |
| **Method** | **Description** |
| ProcessFiles() | Examine the processing queue. Process the file and update the database. |
| CheckForFiles() | Look for files in the folders and place them in the processing queue. |

Here is a portion of the ProcessFiles Method:



To separate the logic and make sure there are no threading problems, the file processing requests are placed into a FIFO queue (CoilFileQueue) by CheckForFiles, and pulled from the queue by ProcessFiles. This is all done within a lock on the queue.

For each coil file processed there is a single master record created in Coils and a number of child records – one for each reading – in CoilReadings.

The readings are actually stored in two different ways:

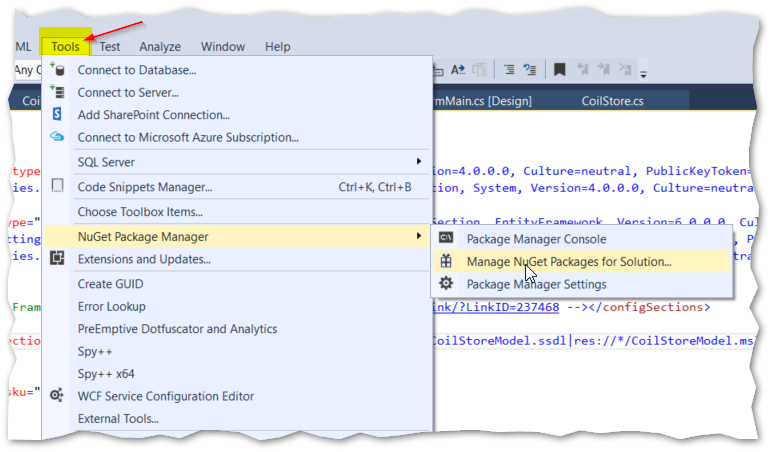
1. Relationally, with one record per reading, and
2. In a comma list within the master record.

There are several reasons for doing this, here are a few:

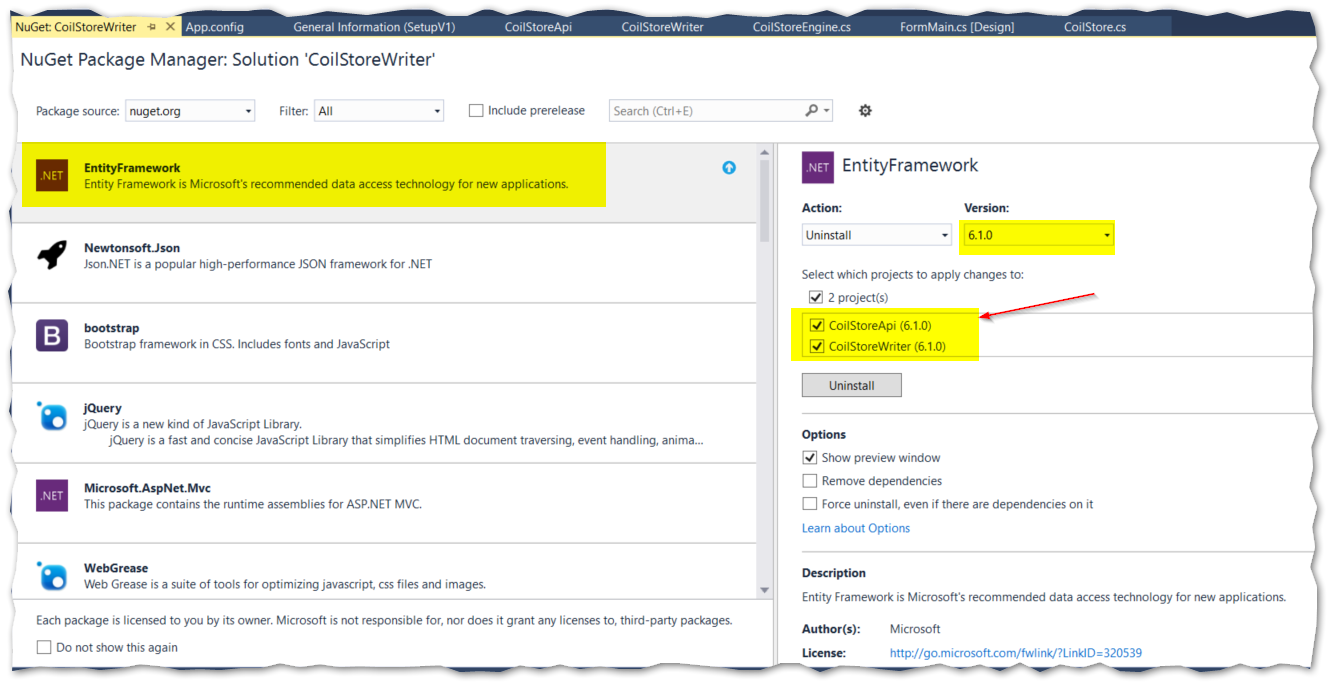
1. By making it relational, it will be easier to use with some off-the-shelf packages.
2. By storing them in the master record, it will be more efficient when a high-volume/capacity application is needed.
3. Since they are within the master, the child records can always be re-created.

## Database Access

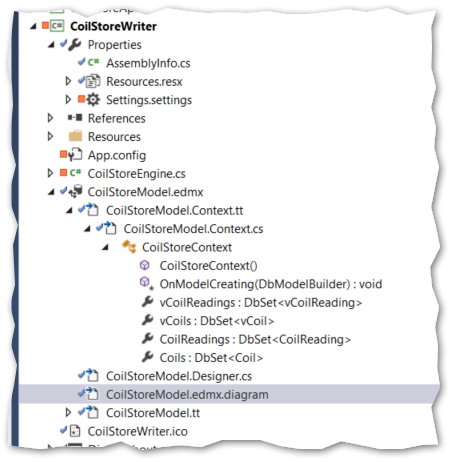
Database access is accomplished with Entity Framework. The version can be checked with NuGet Package Manager:



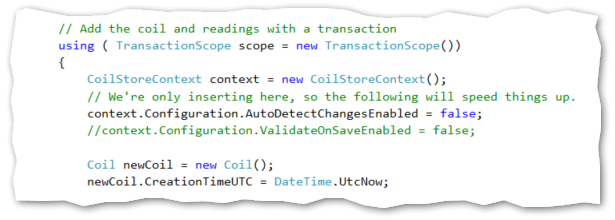
Which brings up something like this:



The created model used in the program is found with the .EDMX extension. The use of Entity Framework is beyond the scope of this manual, but here you can see a partial expansion of the file:



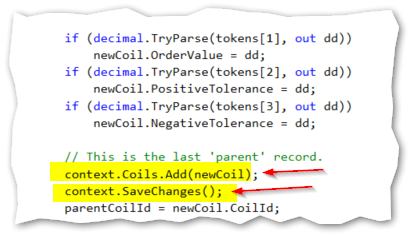
The EF context created is CoilStoreContext. Below are two snippets that show how the transaction scope is created as well as the context being created, as well as a new coil record being created, and then its fields being filled in.



A new record is created in the database by adding this Coil to the Coils collection in the context. This coils collection is the DbSet of type Coil created in the EDMX file. The SaveChanges will place the coil into the SQL table, but within the scope of the transaction.

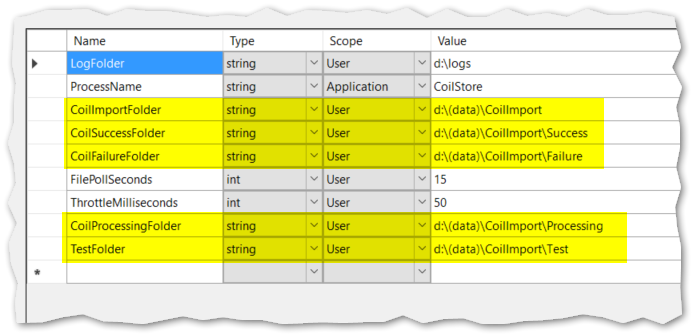
After the Coil record is created, the CoilReading record is created in a similar fashion. This record is also within the same transaction scope, so that if any errors occur, both records written within the transaction are rolled back.

This helps to prevent orphan Coil or CoilReadings records.



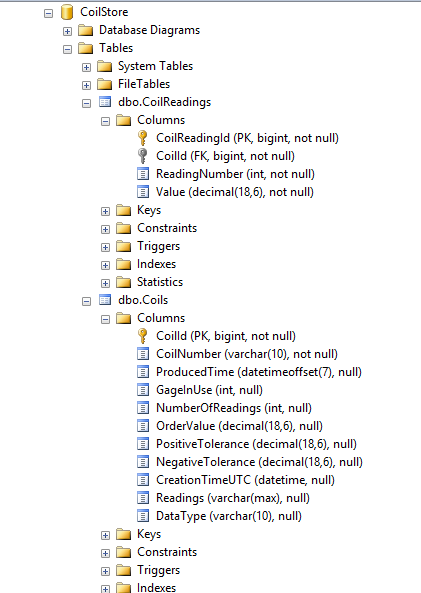
# Appendix – Typical Folder Structure

A typical structure has a parent folder (such as CoilImport) with sub-folders that include Success, Failure, Processing, and Test. The location can be set in the Settings file:



# Appendix – Database Structure

The database consists of two tables with a foreign key on the CoilId field. It is intentionally quite simple, so that it might be implemented on any relational database.



Tables and Fields

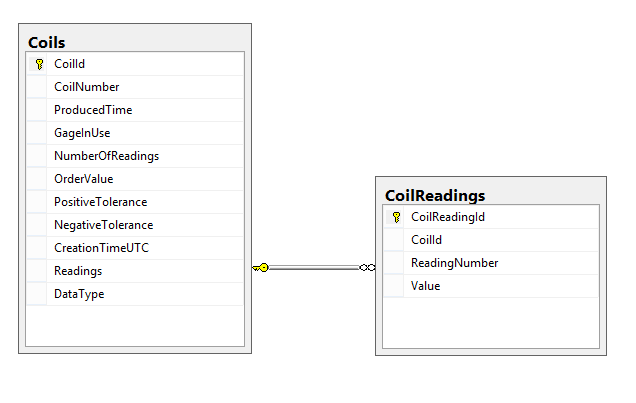
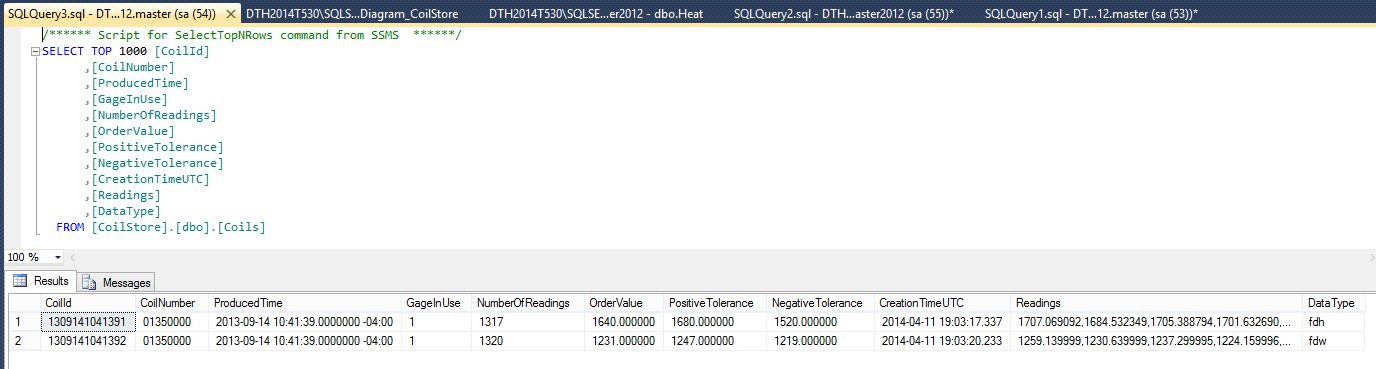
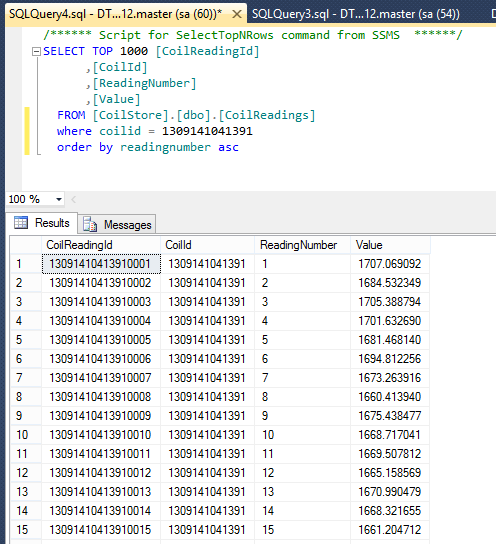


Table Relationship



Sample Coils Data

The child records, with examples:



Sample CoilReading Data

# Appendix - Raw Coil File

The format of the FDH file is an 8 digit name, followed by the extension FDH.

The format of the ASCII data in the file is:

CoilNumber,ProducedDateStamp

GageInUse – (permitted values are 1 and 2)

NumberofReadings – (permitted values are 1 – 2000)

Label1(Actual),Label2(Order),Label3(Pos Tol),Label4(Neg Tol) – (Define the next row of data)

ActualVal1,OrderVal,PosTolVal,NegTolVal

ActualVal2

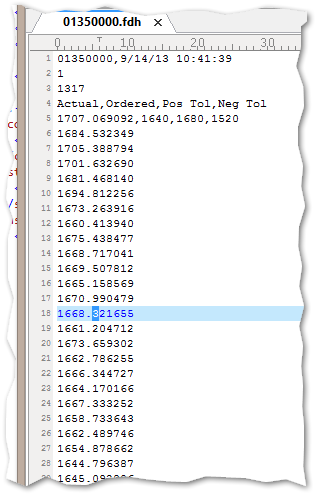
ActualVal3

… ActualVal(n-2)

ActualVal(n-1)

ActualVal(n)

An example of the contents is:



Example Received ASCII Data File

# Appendix – Using Entity Framework and EDMX in Visual Studio

EF and the EDMX file is a large body of information that cannot be covered here. But perhaps a quick explanation on how to alter the database model used by CoilStore would be useful.

The whole idea of EF is to provide a model of the database that can be used by a programmer. This involves a lot of generated code and a very useful but large library of routines.

The place to look for much of this is the .EDMX file in the project folder. If you double-click on this, you are taken to a designer. From here you can right-click to get operations such as the ability to re-generate the model from the database.

