# Overview

In the Energy Imbalance Market During normal operations the software supporting them creates temporary files, working files and processing information onto a Server known as REPOMAN. There are REPOMAN file servers for each environment, Test, QA and Prod. These servers quickly fill with working data and need to be systematically cleared of old, no longer needed files. FileArchiver was designed with this purpose in mind.

# Examples and other details

## Code storage location

The code is stored in TFS at $/ETRMSDev/Utilities/FileArchiver. In this solution is code for the FileArchiver, FileDecompressor and FolderCounter; most importantly is the FileArchiver the others were just support apps that were used in testing and might be useful to you in the future.

## Deployment Strategy

Acquire retention specifications from customer. You will need to know where to place these files, number of days before they are compressed and/or moved or deleted and how long you are required to keep the archived files around before they are deleted. (See: Appendix

Current Folders under EIM on REPOMAN)

1. To deploy build the application as a Release.
2. Copy the 5 files to the target location on your file server. The files are: FileArchiver.exe, FileArchiver.exe.config, FileArchiver.pdb, log4net.dll, log4net.xml.
3. Ensure the configuration file (FileArchiver.exe.config) is pointing to the correct database for log4net output and that log file size and counts are set as desired (currently 25M and 5 files).
4. Create a Task Scheduler entry that uses the normal run command line. (See: Normal usage)

## Deployment example script

Although FileArchiver (FA) can be configured in many different ways, it is recommended that FA be installed on each environment (TEST, QA, PROD) so that the error reporting system which FA places information can report accurately within each environment. When defining those configuration entries in the database be sure to use the CNAME for that database cluster (i.e., ETRMSNONSOXPRODSQL01). This CNAME points to a cluster of failover servers (SQEWPETSQ01V04A, SQEWPETSQ01V04B).

As of the writing of this document it was unknown of any segregation of SOX type files. All files seem to be located on one File Server (REPOMAN) for each environment (TEST, QA, PROD).

In TFS:

* For table build script see “$/ETRMSDev/Utilities/FileArchiver/table build for etrm\_support.sql”
* For table load scripts see “$/ETRMSDev/Utilities/FileArchiver/sql loader for PROD etrmsrepomanprod01.sql” This script is a starting point. The retention of the archived files is 5 years and the retention of the original files is ½ a year. These need to be adjusted depending upon what you find from the customers.

## Command line examples

### Deployment Model Preferred

This deployment model is the desired approach approved by the ETRMS team.

#### Normal usage:

Deployment will be in the three(3) environments (test, QA, prod).FileArchiver should be place on the REPOMAN servers in each environment at the desired drive and folder.(i.e., c:\ETRMS\_Apps\FileArchiver\)

C:\ETRMS\_Apps\FileArchiver\FileArchiver.exe -D:ETRMSNONSOXPRODSQL01 –F:VLSWPETRA01V09–R

C:\ETRMS\_Apps\FileArchiver\FileArchiver.exe -D:ETRMSNONSOXQAQL01 –F:sqewqetra01v09–R

C:\ETRMS\_Apps\FileArchiver\FileArchiver.exe -D:ETRMSNONSOXTESTSQL01 –F:SQEWTETRA01V09 –R

Note: the server name is used instead of the CNAME; can access the server from the server with the CNAME.

### Deployment Model for test and configure

Here are some examples of FileArchiver deployed to the File Server (i.e., REPOMAN). The –F: flag if not supplied but internally FileArchiver assumes the name of the server it sits on.

#### Normal usage:

* C:\ETRMS\_Apps\FileArchiver\FileArchiver.exe -D:ETRMSNONSOXPRODSQL01 –R

#### Test mode:

Here is an example of how to put the app in test mode for checking configuration and what will happen when it is run

* C:\ETRMS\_Apps\FileArchiver\FileArchiver.exe -D:ETRMSNONSOXPRODSQL01 -T

#### Build Mode:

When new folders are added to the configuration database the following will automatically build them for you.

* C:\ETRMS\_Apps\FileArchiver\FileArchiver.exe -D:ETRMSNONSOXPRODSQL01 -T –B

### Deployment Model universal

This deployment gives FileArchiver more latitude to run from a server other than the individual file server it is managing, in other words it can be centrally located on an app server and manage all environment and all file servers within those environments.

#### Normal usage:

When running from an App server across all environments (test, QA, prod) the SourceFolderPath and DestinationFolderPath must user the proper UNC path with the \\servername\share\...

C:\ETRMS\_Apps\FileArchiver\FileArchiver.exe -D:ETRMSNONSOXPRODSQL01 –F:APPSERVERNAME –R

# Design

In designing this application, attention to performance, ease of use, and flexibility was considered.

What was derived was a flexible application that could be centrally located on one server or deployed across different environments. The most optimal approach of deployment would be placing a copy of the FileArchiver on the same server where the files are stored. But if only one installation is desired, the database table of configuration values can be loaded for as many file servers as desired. For performance considerations the contemporary method of Parallel Processing was implemented to take advantage of the current solid state database drives that allow multiple seeks simultaneously; no heads and spinning plate to slow down access.

The parallel processing take place in a method inside the File Management class. The method “ProcessSingleFolderofFiles” is launched in a separate thread for each source folder found in the SQL database. In the method ProcessFilesInParallel the threads are all setup and launched. The method doesn’t release control back to the application until all threads have completed their tasks, this is accomplished by the thread Join function.

In processing the files, the dates of moved or compressed files are by default the current date and time. It was decided to set these new file dates back to the original date and time of the original file for a consistency purposes. The files are named with a date and time component separated but an underscore “\_” along with other identifying name components (i.e., BA\_Tags\_UCS220181006\_051032\_33120887). The format is as follows “YYYYMMDD\_HHmmss”. This date format is embedded within the file name. So a simple Regular express is used to identify and locate the date time part and then set the newly compressed or newly moved file to this date. The application will only delete the original file after a successful transfer to the new location.

## Command Line control

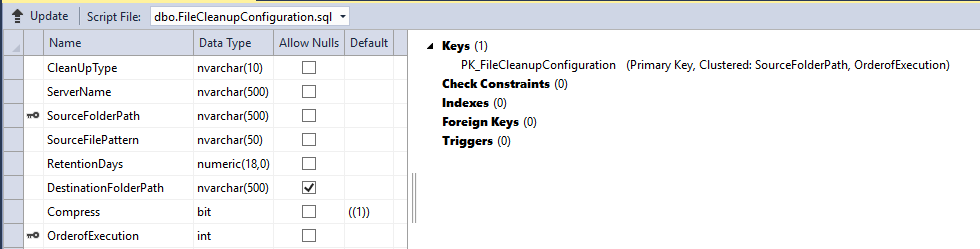
A robust set of command line controls have been created to manage the files on the various servers. There is both a run and a test mode. In the test mode you will discover if the folders both source and target exist. The system will alert you of these issues. There is currently an additional flag that can be used in conjunction with the –T flag; and that is –B flag. FileArchiver will use this flag to create the directories found in the configuration database?

Caution: the create directory process does not perform any checks on the path naming convention, it simply looks for it, if it doesn’t exist, it creates it. Use it carefully.

|  |  |
| --- | --- |
| Flag | Description |
| -D: | Followed by the Database server name is where the configurations are stored for all. |
| -F: | Followed by the File Server name where files to be managed are located. |
| -R | This flag enables the application to run normally; without it the app won't run. |
| -T | This flag tests the command line arguments and configuration values. The R and T flag are mutually exclusive, if both are used the T flag shall take precedent |
| -B | This flag in conjunction with the -T flag will build the directory if it does not exist. |

## Database Details

The database portion of FileArchiver consist of a single table “FileCleanupConfiguration”. The purpose of this table is to allow the configuration/maintenance tech to tell FileArchiver what folders and files to manage.



The following is a description of what each field in the table does.

### CleanUpType

Two possible values for this field are Archive and Delete.

* Archive - will move and/or compress the files whose create date is older than the current date minus the RetentionDays value. Setting the values to 0 will essential move or compress all files.
* Delete – will remove the files whose create date is older than the current date minus the RetentionDays value.

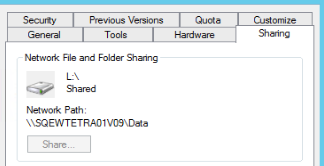
### ServerName

This field holds the name of the File Server where the data files reside. There are two types of Server names that can be applied depending upon where FileArchiver is installed; those types are CNAMES (Alias) or the DNS Server Name.

If FileArchiver is installed on the File Server then the DNS name is required. In the rare exception where the FileArchiver is installed on one file server but the data files are on another server, then the CNAME is required.

### SourceFolderPath

This field, depending upon the file server configuration, contains the UNC path to the folder where the data files are located that need managing. The optimal approach is to have a share name to the High level folder or drive where these group of files reside. (i.e., Data share for L: ).



If a share is not available, then try [\\ServerName\L$](file:///\\ServerName\L$) if running remotely or if running locally then the Drive and Path will work (i.e., L:\EIM\Interfaces\LoadOffice\SystemLoad\PB\_250). Within FileArchiver, all paths are programmatically suffixed with a backslash ‘\’ if the configuration was not set that way.

### SourceFilePattern

This field is for controlling what file types or file names the FileArchiver works with. Typically the value for this field will be ‘\*.\*’ indicating all files. But if the need to look at different file types (i.e., \*.csv, \*.xml) arises then this filter can manage it. Also in the case where something like the WebTraderNetDeals/PB\_249\_252 where different files of the same type, like \*.csv but several file class names like DayAheadDealHourly, RealtimeDealsHourly, GenerationDealsHourly then one can construct a filter for each file class, making a separate configuration record for each.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CleanupType | ServerName | SourceFolderPath UNC | SourceFilePattern | RetentionDays | DestinationFolderPath UNC | Compress | Execution order |
| Archive | ServerName | UNC Source Path | DayAheadDealHourly\*.\* | 5 | UNC dest Path | False | 1 |
| Archive | ServerName | UNC Source Path | RealtimeDealsHourly \*.\* | 15 | UNC dest Path | False | 2 |
| Archive | ServerName | UNC Source Path | GenerationDealsHourly \*.\* | 100 | UNC dest Path | False | 3 |
|  |  |  |  |  |  |  |  |

Note: when working in the same folder with multiple file types, classes, etc. it is important to provide a unique execution order number. Since same source path processing is handle in one thread, we don’t want any collisions of same file in different thread; although with the file name SourceFilePattern filtering on unique classes this probably won’t be a factor.

At some point you are going to want to delete the compressed archived files as well. A configuration can be added that will govern how this is executed.

Example: To delete compressed files after N days from UNC Path on Server Name, order of exe M configure like this:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CleanupType | ServerName | SourceFolderPath UNC | SourceFilePattern | RetentionDays | DestinationFolderPath UNC | Compress | Execution order |
| Delete | ServerName | UNC Path | \*.gz | N | Null | False | M |

### RetentionDays

This field controls how long to wait before a file is processed. It currently only works for a day count. Future enhancements might consider end-of-month or other types of temporal classifications to determine what files to process.

### DestinationFolderPath

This field provides the target location to where the processed files will go. UNC or Share name or Local directory can all be used in this field. The application will test these settings and fail the run if all applicable paths are not set correctly or if a path has been removed from the file server.

If a Null value is used then FileArchiver assumes the same folder as the SourceFolderPath

### Compress

This is a 1, 0(true, false) flag that tells the application to compress the files into a gzip format. Once compressed and successfully saved to disk the source file will be deleted. There is a decompression app for your use within this project in case you want to decompress the file(s).

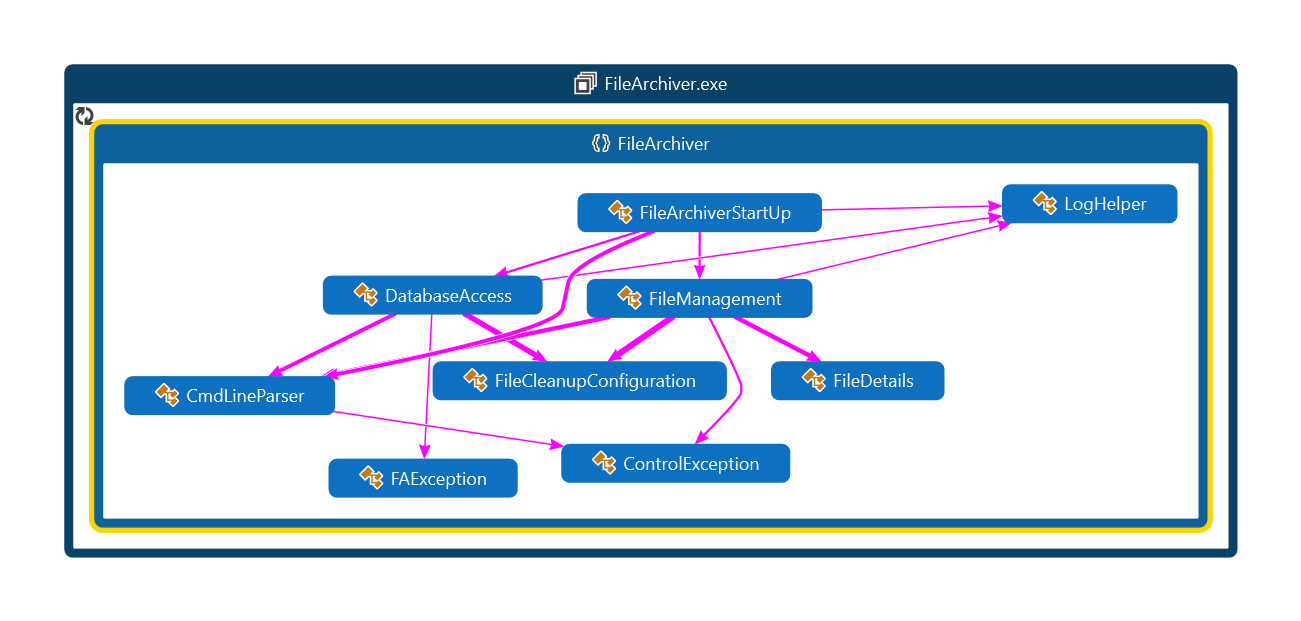
### OrderofExecution

This field is provided for the case where multiple configuration records having the same source file path. FileArchiver will find all records with same source file paths and place them into one thread, at which point it will be necessary to put order to their processing.

## Application Class design

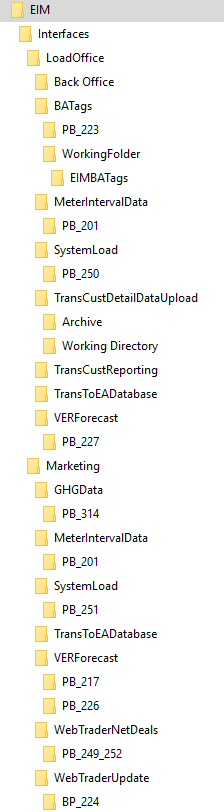
The application was developed with a separation of duties in mind. Database kinds of things were in the DatabaseAccess class. File management kinds of thing were in the FileManagement class.

Several helper classes were produced to help in various things like command line processing, identifying column names, special error catching situations and log4net helper class.

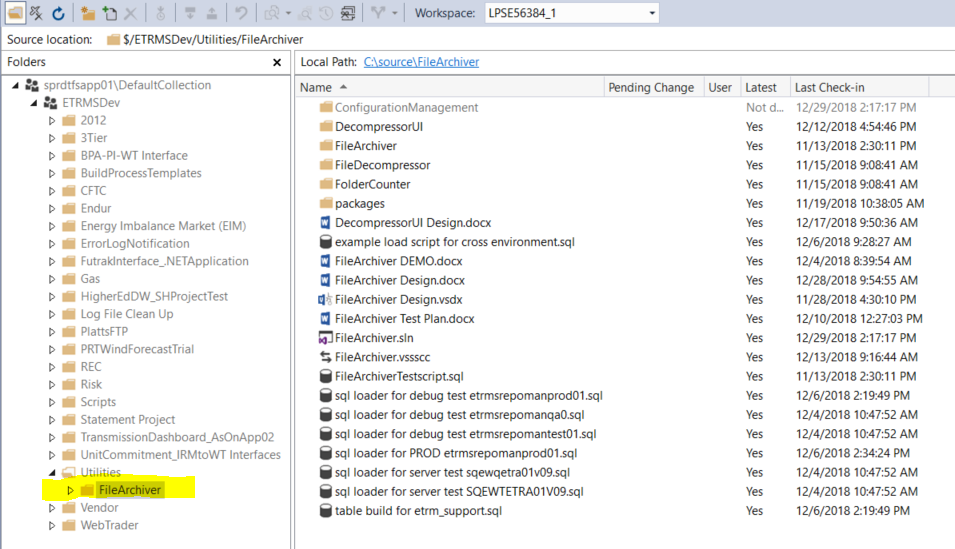


# Appendix

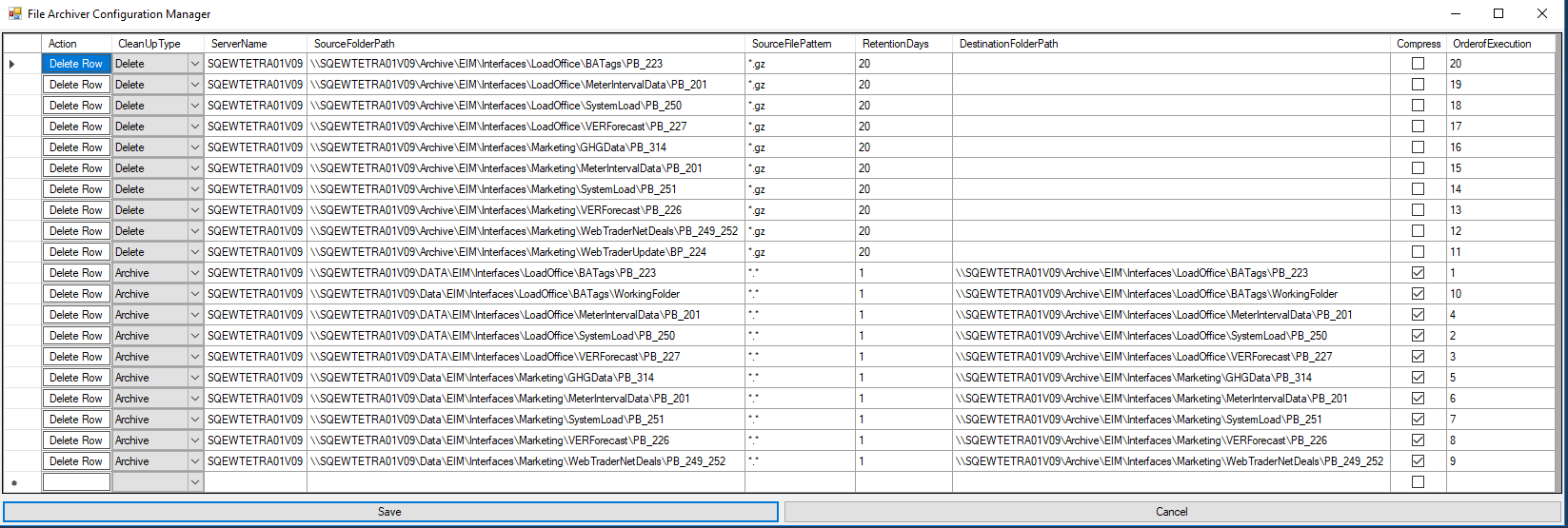
## Current Folders under EIM on REPOMAN



## TFS location:



## ConfigurationManagement



This tool allows easier access to the database table controlling the FileArchiver. It allows new rows or deletion of existing rows.

Note: the application looks at the server that is found in its ConfigurationManagement.exe.config file. In there you can set both server and database any other connection parameters that will suite your needs.

Note: the Save and cancel buttons on the bottom. Press Save to save any changes. If you press the delete Row button under the Action column you must also press save to commit your changes. Cancel will Rollback any changes and close the screen.

The columns are defined as follows:

Action – Each row has a delete button and will delete that row when press, but the change will not occur in the database till the Save button is pressed.

CleanUpType – A drop down list with two choices, Archive and Delete. This controls the FileArchiver’s processing. Delete will remove the file. Archive will move the file to a designated location found in DestinationFolderPath column. If Compress is checked the file will be compress in a GZip format before it is moved to the designated spot. If the DestinationFolderPath is empty the compression will occur at the SourceFolderPath and the original file will be removed.

ServerName – This column provides FileArchiver with a means to filter the table to access only the records needed for FileArchiver’s operation. In theory one could put several different server names and/or CNAMES here to have a “one stop shopping” location for configuration.

SourceFolderPath – this is usually the UNC path for where to find the files to be archived. In normal operations the FileArchiver will be on the machine that houses the business files. The contents of this column in this case can be just the Drive letter, colon, backslash, then the rest of the path (is that considered UNC too? Depends on who you talk to but the strict definition is \\server-name\shared-resource-pathname).

SourceFilePattern – is the location where you can control what kind of files are processed.

RetentionDays – is the number of days delayed before FileArchiver processes the files.

DestinationFolderPath - is the path where the files will be placed for long term storage. Usually a different drive but this could be a completely different server and drive (UNC). Beware however, performance suffers when copying across the network.

Compress - Tell File archiver whether or not to compress the files being archived.

OrderofExection – is a column that is needed for special cases where SourceFolderPath is the same. The reason we do that is because we might want to process different file names or file types in the same folder. Best practices for this column is to make sure the numbers don’t repeat. This column was added for those cases where users may have become sloppy in their configurations and the process tried to work on the same file twice in this multithreaded environment.

## Archiver Configurations

The FileArchiver (FA) can be configured in many different ways. The following diagram highlights the possible configurations available.



The four configurations are labeled A-D and the details are as follows:

1. This configuration moves/compresses files on the same server. Different drives can be utilized and typically there is a Data drive and an Archive drive. This is the most efficient and quickest way to manage files.
2. For moving/compressing files to a workstation or laptop. Note: the FileArchiver is typically launched from a task scheduler so the target workstation must be on and available on the network at the scheduled time of the scheduled task.
3. This configuration is like B but just a different file server at the data center.
4. FileArchiver can live on an application server separate from the server holding the business files. However, this configuration will run slower than the other configurations because of the network traffic/IO latency. If this arrangement proves sufficient operating speeds, in theory, you could operate all environments from one instance of FileArchiver. Given the threading nature of the application with all environments you may experience a minor slowing because the cores on the processor may overload but it would take a lot of folders in the configuration database to slow it significantly.