Quartz Server Installation.

Introduction

Spans project utilizes the Quartz scheduler – open source job scheduling framework.  
  
Spans project is organized as a set of assemblies, with business logic located in the SpansLib.dll, it depends on another TradeData assembly – SharedLibrary.dll, which contains a set of helpers. In order to run the code I use jobs, job is a class with some logic, inherited from the IJob interface, which is defined in the Quartz.dll. This job then must be registered in the Quartz server. The scheduler (Quartz) should be installed as a windows service, properly configured and started. It then will start jobs according to their configuration and schedules, which in this particular case are completely defined in a xml file - quartz\_jobs.xml.  
  
Actually all settings required for proper functioning of the Quartz server are already defined and stored in the Spans project, you only need to download / unpack the Quartz server archive and copy project files to the installation directory.

The step-by-step instruction is below, and after it there is a reference with description of project files and jobs / server configuration settings.

Files of the project are located in 3 repositories, you will need to create a local copies, build solution and copy binaries to the scheduler job’s folder.

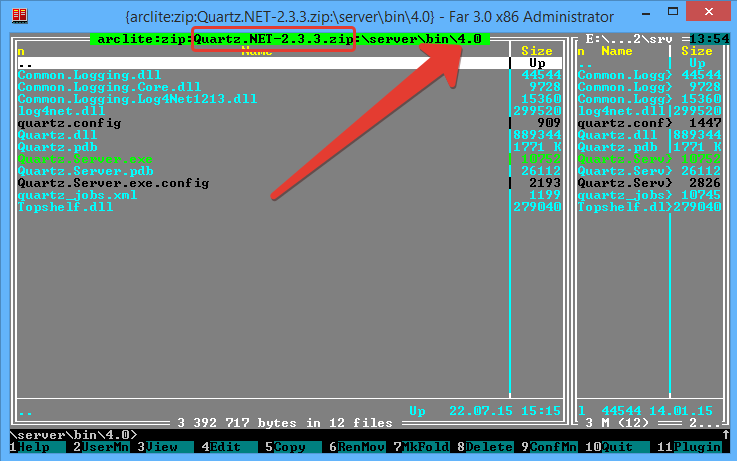
Installation

You may choose semi-automated scenario or manual, the difference is that in an automated one, all steps are defined in the batch file, but some interaction will be required.

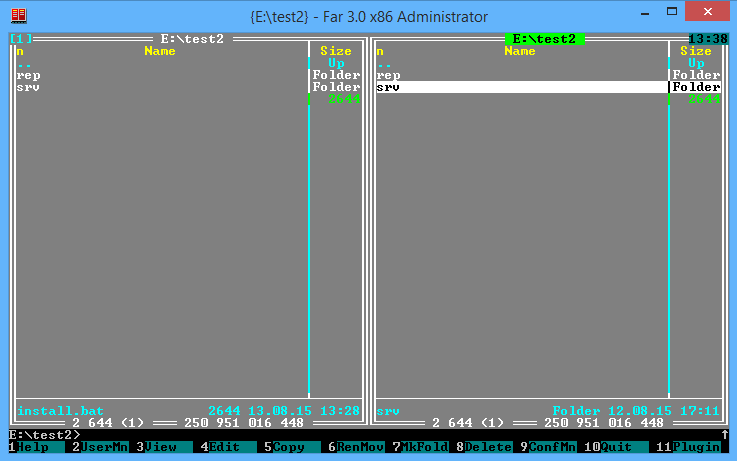
1. Prerequisites   
     
   1. Install the SqlXml 4.0 SP1 (or later) from <http://www.microsoft.com/en-us/download/details.aspx?id=30403>  
   It is used for bulk insert from xml files into database.  
     
   2. Verify you have the Atlassian SourceTree installed on the machine. You can download it here: <https://www.sourcetreeapp.com/>

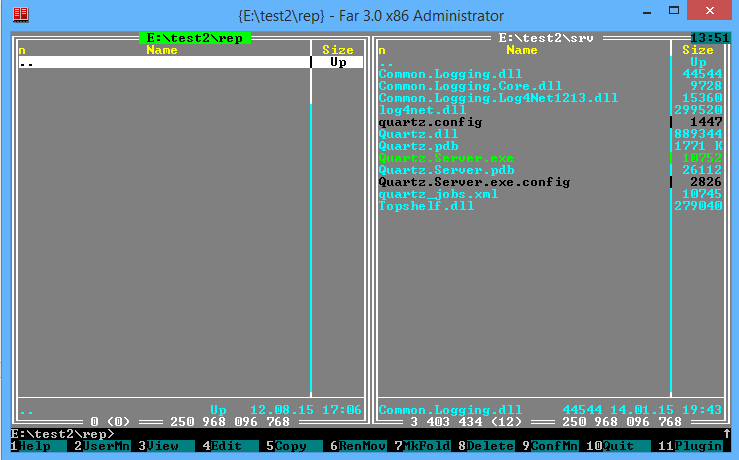
3. SQL Server Management Objects (SMO). If you do not have them, you can download a feature pack, for example for SQL Server 2008 R2: <http://www.microsoft.com/en-US/download/details.aspx?id=16978>

4. Download latest version of the Quartz server archive at <http://sourceforge.net/projects/quartznet/files/quartznet/>  
From this archive you need binaries from the **server\bin\4.0** folder inside of a zip file.

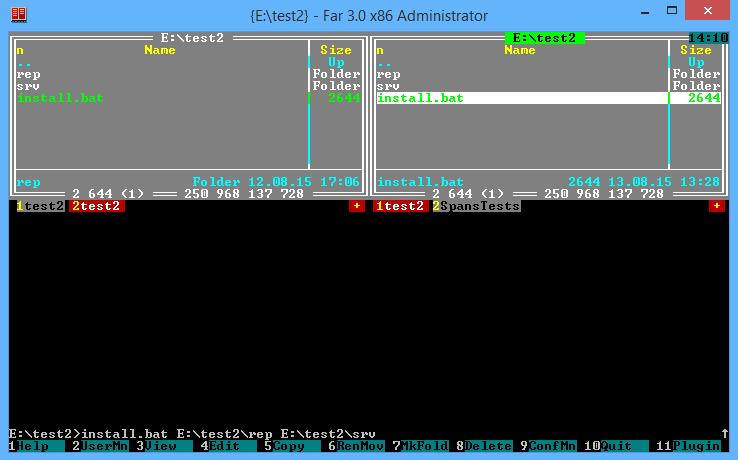


Copy files from the archive to a location, where you want the Quartz server instance.

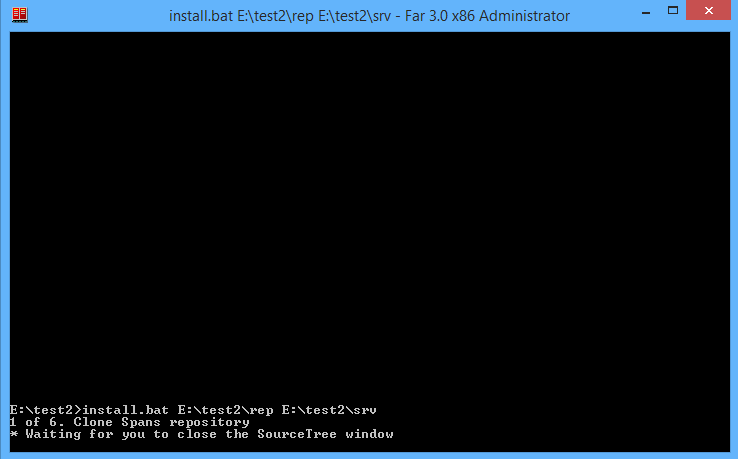
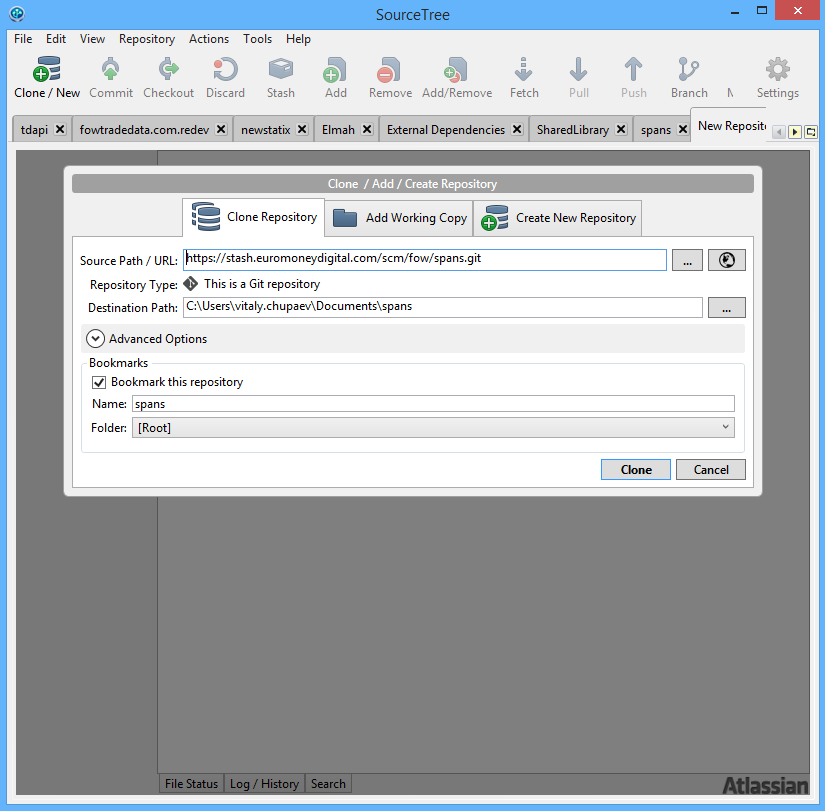
1. Semi-automated installation  
     
   1. Create a root folder for repository files and another one for a Quartz server files. The installation batch file is located in the same directory with this document, which you are reading now, the relative path in the repository \spans\Installation\install.bat  
     
   **Install.bat** expects two parameters –   
    **first** – repositories root folder  
    **second** – Quartz server root folder  
     
   There will be three repositories, which must be stored on the same level.  
   For example I have a **e:\test2** folder with subfolders **rep** for repositories and **srv** for server  
   

And inside subfolders (before you start, the repositories root folder must be empty, the server folder must contain quartz files)  


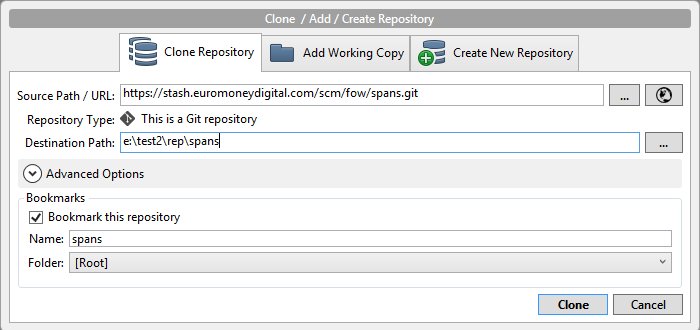
2. Run install.bat. There will be 6 steps, first three will require interaction, they will ask about the path where to clone repositories and you will need to close SourceTree window after each step manually.  
  
! Close all opened SourceTree windows before start, otherwise batch will not wait until repositories are cloned and fail.  
  
Run batch like this: install.bat e:\test2\rep e:\test2\srv



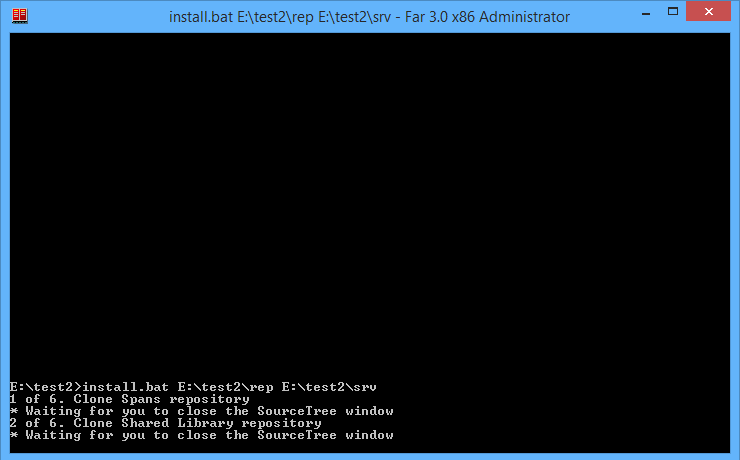
Do not use quotes around the paths.

Step 1  
  
  
Once you’ve started, you will see a dialog box of the SourceTree client, the URL is already provided from the batch file, but local path cannot be specified from a command line, at least in the current version, and you should set it to match root path provided as a parameter.  
 **Spans** repository **must** have **spans** local folder  


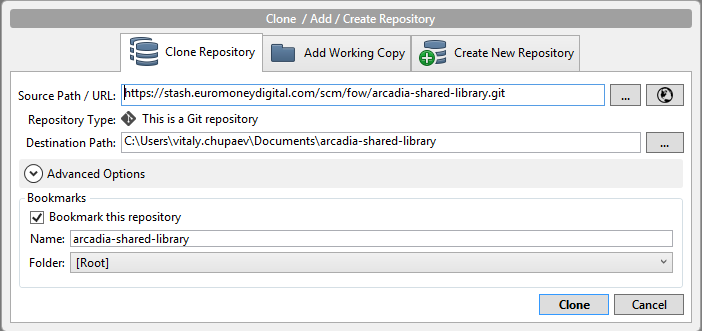
Change path:

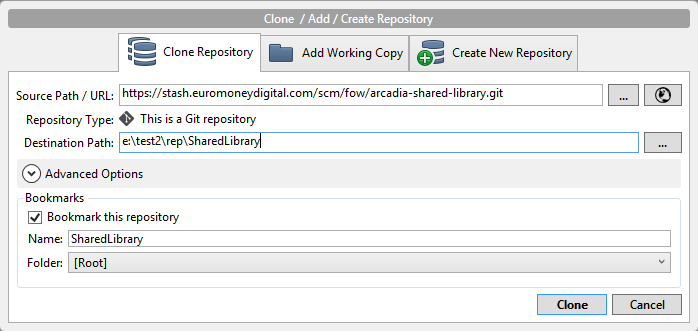


And click Clone button, once it is finished, close dialog and SourceTree window.

Step 2  


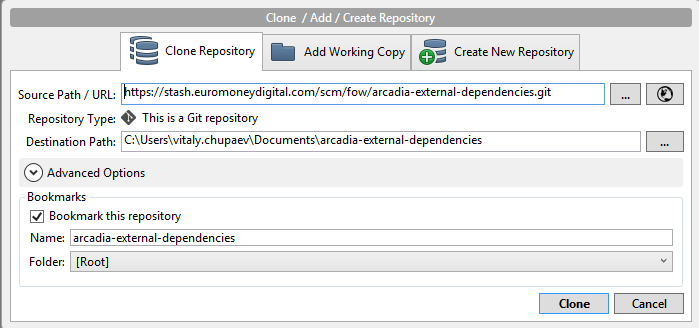
**Arcadia-shared-library** rep **must** have **SharedLibrary** local name.



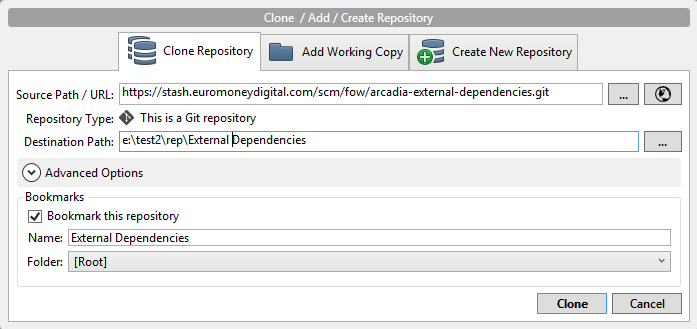
Change to:  
  


All the same – clone and close.  
  
Step 3

And finally **arcadia-external-dependencies must** be an **External Dependencies with whitespace.**When you type in SourceTree dialog it won’t let you insert a whitespace until there are some symbols on the right, **so first type ExternalDependencies, and then add a whitespace**

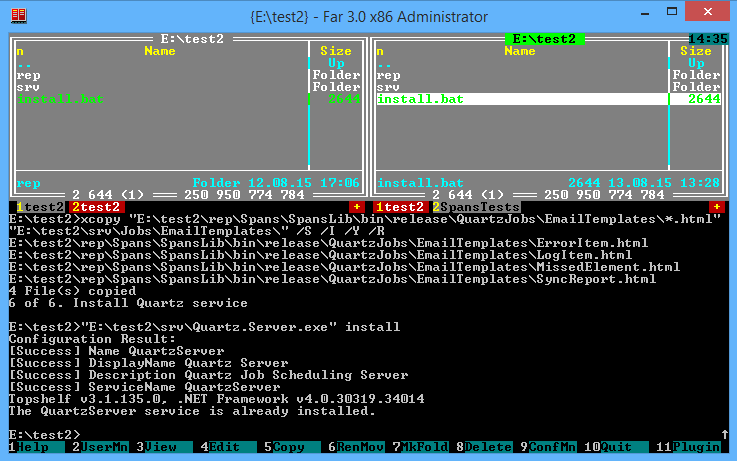


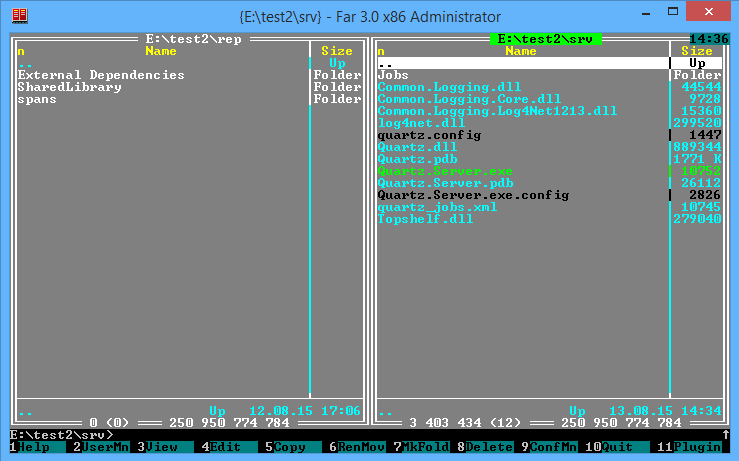
Change to:



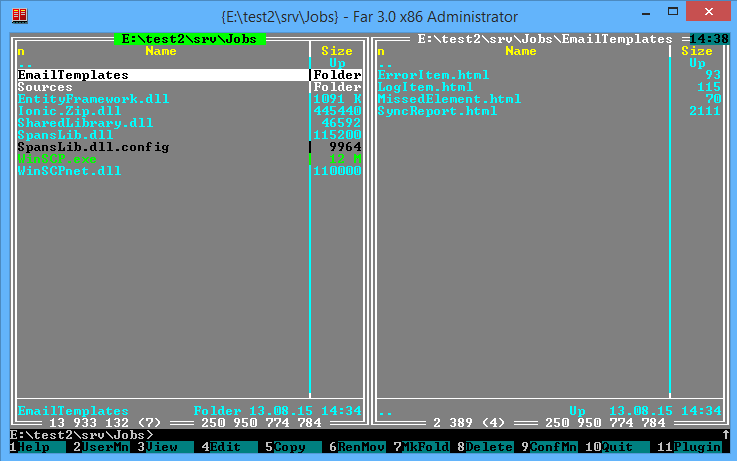
Clone & close.  
  
Step 4, no interaction  
After that, the msbuild will be started, the path will be collected from the registry.  
During this step, msbuild will try to download NuGet packages which are used in the project, so you need the machine to be connected to the Internet.  
  
Step 5, no interaction  
Then once it is finished, files will be copied from the build path to the server folder.

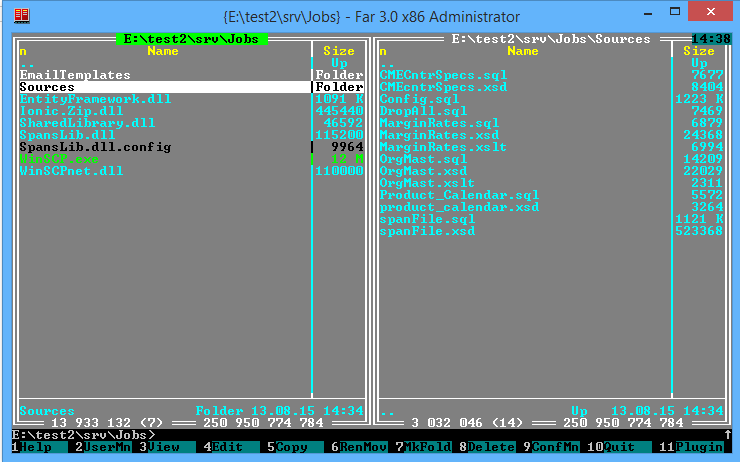
Step 6, no interaction  
Last step is to install the Quartz as a windows service.

Done:

Now folders look like this:  


Notice that e:\test2\rep contains all three repositories, and e:\test2\srv contains a new folder Jobs

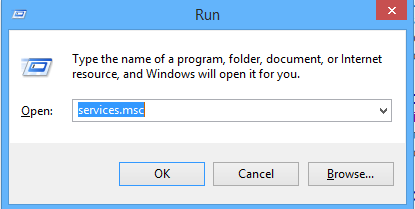
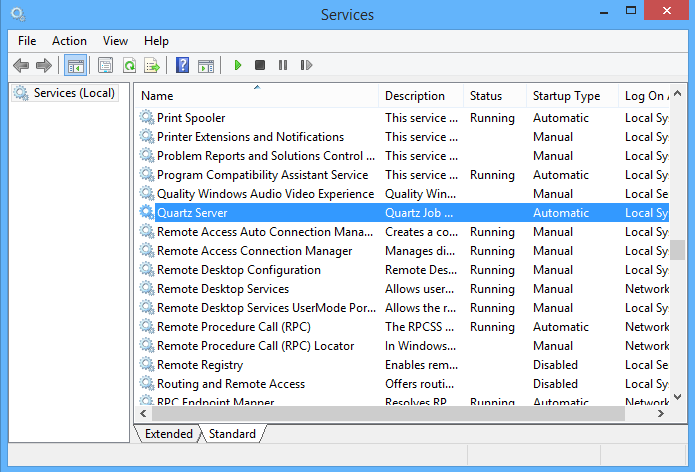
Inside Jobs:  
  




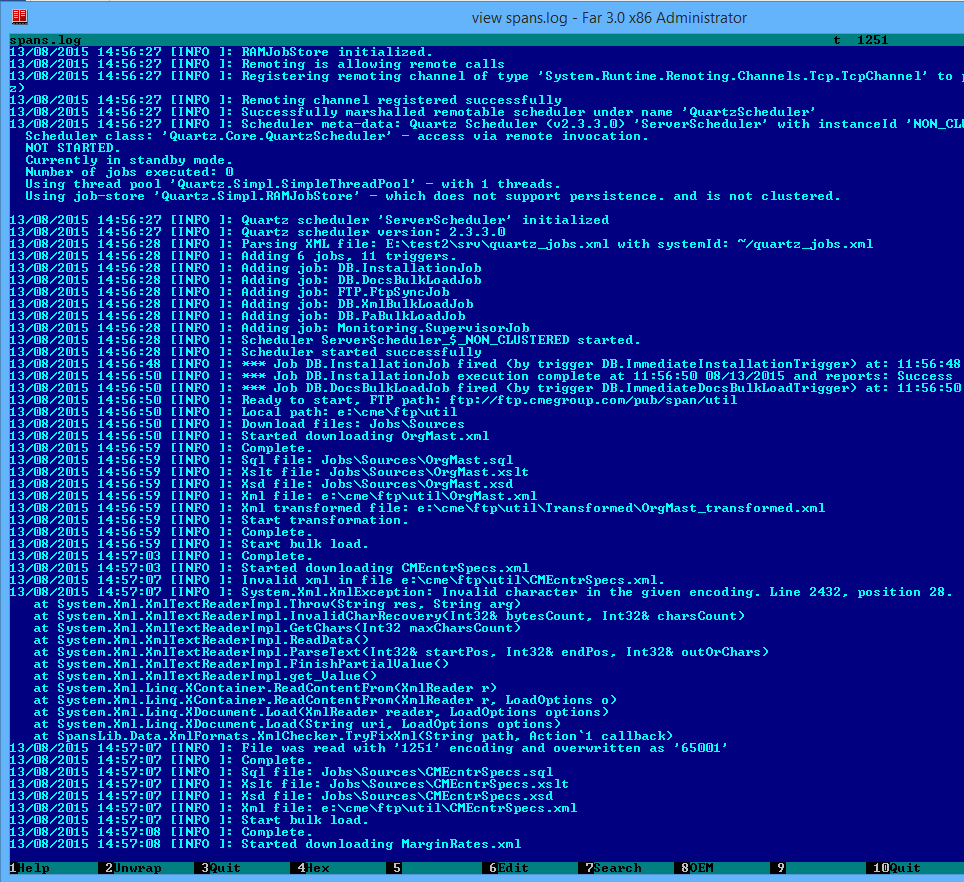
If you have issues during installation, you can start it once again and redirect all output to the file, like this  
**install.bat e:\test2\rep e:\test2\srv >res.txt**

3. Now you need to review   
 - E:\test2\srv\quartz\_jobs.xml (you work with your own paths and I just use mine as an example)  
 - E:\test2\srv\Jobs\SpansLib.dll.config

Check connection strings, smtp parameters and local paths, change them to match your environment (see the settings description at the end of this document).

4. Start the Quartz service, press Win+R, type services.msc  
  
  
  


Get back to the e:\test2\srv folder, notice a new one – Logs subfolder, open the spans.log file to check the service’s healthiness.



1. Manual installation  
   This is another possibility, you may prefer it mostly if you want to review the code.  
   You must have pass the prerequisites steps already.
2. Open SourceTree and clone 3 repositories:  
   <https://stash.euromoneydigital.com/scm/fow/spans.git>  
   <https://stash.euromoneydigital.com/scm/fow/arcadia-shared-library.git>  
   <https://stash.euromoneydigital.com/scm/fow/arcadia-external-dependencies.git>  
     
   They all must be on the same level, for example   
   E:\test2\rep\spans  
   E:\test2\rep\SharedLibrary  
   E:\test2\rep\External Dependencies
3. Open E:\test2\rep\spans\SpansUI\SpansUI.sln in Visual Studio 2012 or later. Check references and build the solution.  
   Projects use assemblies from SharedLibrary, External Dependencies, you must have SqlXml 4.0 and SQL Server Management Objects installed.
4. Now you need to review   
   - E:\test2\rep\spans\SpansLib\quartz\_jobs.xml   
   - E:\test2\rep\spans\SpansUI\App.config (this config also added to the SpansLib project as a link, so there is one file for two projects)  
   Check connection strings, smtp parameters and local paths, other settings (see the settings description at the end of this document)
5. - Create a new subfolder **Jobs** in the folder where the Quartz server files are located, in my case E:\test2\srv,so that you have e:\test2\srv\Jobs and, copy output files (without folders):  
   **from** E:\test2\rep\spans\SpansLib\bin\Release\\*.\*  
   **to** e:\test2\srv\Jobs  
   - Create E:\test2\srv\Jobs\EmailTemplates, copy   
   **from** E:\test2\rep\spans\SpansLib\bin\Release\QuartzJobs\EmailTemplates\\*.\*  
   **to** E:\test2\srv\Jobs\EmailTemplates  
     
   - Create E:\test2\srv\Jobs\Sources, copy  
   **from** E:\test2\rep\spans\Sources\\*.\*  
   **to** E:\test2\srv\Jobs\Sources  
     
   - Copy files with replacement:  
   E:\test2\rep\spans\SpansLib\quartz.config **to**  E:\test2\srv\quartz.config  
   E:\test2\rep\spans\SpansLib\Quartz.Server.exe.config **to**  E:\test2\srv\Quartz.Server.exe.config  
   E:\test2\rep\spans\SpansLib\quartz\_jobs.xml **to**  E:\test2\srv\quartz\_jobs.xml  
   These files will change the Quartz server behavior  
   quartz.config - 1 thread,  
   quartz\_jobs.xml – jobs ,  
   Quartz.Server.exe.config - logging configuration, probing, useLegacyV2RuntimeActivationPolicy  
     
   - Delete from E:\test2\srv\Jobs:  
   Elmah.dll  
   LukeSkywalker.IPNetwork.dll  
   Newtonsoft.Json.dll  
   Telerik.Web.UI.dll  
   Telerik.Web.UI.xml
6. Open cmd.exe as Administrator and run **Quartz.Server.exe install  
     
   **
7. Open services.msc, find the **Quartz Server** service and start it.
8. Open E:\test2\srv\Logs\spans.log, it is nice to open it in a viewer which updates file automatically and keeps the position at the end of the file, I use the built-in viewer of the Far Manager (<http://www.farmanager.com/index.php?l=en>, it’s from the same developer as RAR archiver).

Reference

Quartz scheduler tutorial can be found here <http://www.quartz-scheduler.net/documentation/quartz-2.x/tutorial/index.html>

# Jobs.

* Job is a class inherited from an IJob interface, it contains an Execute method.  
  The parameters could be passed to this method.
* Some configuration parameters are kept in the SpansLib.dll.config, but some are passed to the job method and duplicates the app.config, e.g. connection string. This is done for flexibility, as there are several jobs, each of which may have many triggers, and at some point you may want to have different values for same parameters.
* Jobs schedule and parameters are defined in the quartz\_jobs.xml file.  
  There is a <job-data-map> structure which allows to configure job parameters, it can be specified in both – job and trigger, the trigger’s parameters have precedence over job’s parameters.
* There is no description of quartz\_jobs.xml but there is an xsd file comes with Quartz installation, and IntelliSence in Visual Studio shows suggestions, when you edit it. While it is useful, the elements in xsd defined as sequence, which means the order does matter, so IntelliSence will show a different set of elements on the same level, depending after which element you decided to add a new one.

Jobs currently have two triggers for each, one simple trigger to start in one minute after server start (1 minute is a time to attach project to the server for debug) and one cron trigger to run at 1:01 AM every week day. Only the DocsBulkUploadJob has a cron expression to run every first day of the month.

# Jobs descriptions and parameters

Currently the Quartz server is configured to have only one thread (**quartz.config**, quartz.threadPool.threadCount = 1), which means that all jobs will run consequently. All jobs have the same start time and priority value. Priority allows to control which job will run first. The job with the lowest value will run last. Minimal priority – “1”, in default configuration has the **SupervisorJob**, it means it will run after all other jobs.

**InstallationJob**, currently configured to run once per service start, it will run installation SQL scripts, if necessary.  
There are two of them - stash\spans\Sources\spanFile.sql – for tables that matches the span files structure and stash\spans\Sources\Config.sql – contains all configuration tables with information about PA (pa2, pa3, pa5, pa6) formats.  
Job has several parameters:  
connStr – connection string to database  
overwrite – scripts files contains instructions to drop all existed objects the script deal with, if overwrite is true, the script will run, otherwise – not.

**DocsBulkLoadJob**, currently configured to run once per month, it will overwrite existing data.  
Each xml file from the Util folder on ftp has its own script, transformation and schema files in the Sources folder of the Spans project.   
Job has several parameters:  
connStr – connection string to database  
ftpPath – folder with files on FTP side  
filePath – where to put xml files  
serviceFiles – folder with sql, xsd, xslt files

**FtpSyncJob**, currently configured to run daily, creates a list of files on FTP server and in local folder, then compares files by name and downloads files which are not present locally,   
parameters:  
connStr – connection string to database  
ftpPath – the starting point on FTP  
localPath – where to put files from FTP, the folders structure will be kept  
skipFolders – list of folders to skip when traversing the ftp server  
skipFileTypes – files with listed extensions will be skipped, separated with semicolon, comparison is done with EndsWith()  
daysBack – files are filtered by date, it uses Today, minus a number of days specified in the parameter, parameter should be a positive number  
overwrite – true to drop all tables of the format and write data from the current data set

**XmlBulkLoadJob**, currently configured to run daily, looks for files with corresponding last write date, unpacks them when necessary and then builds an xml files list also by corresponding date (as it may differ from the archive date, for example the archive is of 30.07.2015 and the file inside is of 29.07.2015, the real data date is a data file’s date).  
If file with the same name has been uploaded, the last write date of the file will be compared to job’s start date, if start date greater than write date, file will be uploaded to db.  
It checks the **spanFile** table to see if the file was uploaded, if it was chosen to append, then the uploaded files will be skipped, parameters:  
connStr – connection string to database  
path – where to look for files   
daysBack – files are filtered by date, it uses Today, minus a number of days specified in the parameter, parameter should be a positive number  
overwrite – true to drop all tables of the format and write data from the current data set

**PaBulkLoadJob**, currently configured to run daily, looks for archives files with corresponding last write date, unpacks them when necessary and then builds a files list of text formats also by corresponding date (as it may differ from the archive date, for example the archive is of 30.07.2015 and the file inside is of 29.07.2015, the real data date is a data file’s date). If file with the same name has been uploaded, the last write date of the file will be compared to job’s start date, if start date greater than write date, file will be uploaded to db.  
It checks the **paFile** table to see if the file was uploaded, if the choice - to append, then the uploaded files will be skipped, parameters:  
connStr – connection string to database  
path – where to look for files   
daysBack – files are filtered by date, it uses Today, minus a number of days specified in the parameter, parameter should be a positive number  
overwrite – true to drop all tables of the format and write data from the current data set

**SuperviserJob**, currently configured to run daily, it runs after all other jobs and looks into **import\_Log**, **import\_Error** tables, calls the **sp\_FindMissedElements** stored procedure to see if there were missed elements during XML import.  
It sends then email notification with the results, no matter success or failure.  
Parameters:  
connStr – connection string to database  
hoursBack – the period for which to take log entries, as it is not possible to pass data between jobs directly. If jobs runs daily it is enough to set several hours, depending on overall jobs duration. By default – 23.

# App.config, AppSettings.tt

The original app.config located in the SpansUI project, the SpansLib project has a link on that one.  
Each project has a T4 template - AppSettigns.tt, which generates the AppSettigns.cs file with parameters accessors.

Templates read appSettings section and contain some logic, during generation there is an attempt to convert data to bool, decimal or int, so if value in app.config was set to “true” or “false” the accessor will be generated as Boolean, similar for integer and decimal, otherwise it will be a string.

If string contains semicolon (;), the accessor will be generated as IEnumerable<string>, split by semicolon.

If parameter name ends with “Template” (case sensitive), semicolon will be ignored and accessor will be of a string type.

AppSettigns.tt in the SpansLib project also has logic to work with the assembly’s config when the assembly is used in the Quartz server environment.

The parameters:  
FtpMirrorPath – the root folder to put directories\files from FTP, it is overridden with job’s parameters, and currently used only in UI.

SpanSchemaFile – path to the spanFile.xsd, the schema file for span xml files.

SpanSqlFile – path to the sql script, which defines a table structure for span format.

SqlXmlBulkLoadErrorFile – when error occurs during bulk load with SqlXml 4.0, this file will contain the error info

FtpTimeoutInMilliseconds – time to wait for response from FTP server

FtpLogin – for anonymous access use “anonymous”

FtpPassword – for anonymous access use “anonymous”

SmtpHost – as it sounds

SmtpPort – as it sounds

EnableSsl – use SSL with smtp

Recipient – as it sounds

Sender – as it sounds

Password – provide password if it’s required for smtp, otherwise leave blank, but do not delete. If password is specified the Sender will be used as login.

SubjectTemplateSuccess – as it sounds

SubjectTemplateFailed – as it sounds

BodyTemplate – as it sounds

ErrorItemTemplate – as it sounds

LogItemTemplate – as it sounds

MissedElementsTemplate – as it sounds

Currently there is a particular problem with CMEcntrSpecs.xml encoding, so next two parameters used for fixing:

XmlReadEncoding – read file with encoding specified in this field

XmlWriteEncoding – and saves it as UTF-8 (65001)

XmlFormats – list of extensions to treat as a span file

TextFormats – list of extensions to treat as pa file

FtpUtilsUrl – ftp path to CMEcntrSpecs.xml, MarginRates.xml, OrgMast.xml, product\_calendar.xml

FtpDataUrl – ftp path to data files root

FtpFiles – list of files to download from Util

FtpNumberOfThreads – number of simultaneous threads for downloading

For PA files:

DefaultStringColumnType – type of string columns in tables generated for data from PA files

FileIdFieldName – Additional field names are defined in the cfg\_ExtraFields table, here it is used as a constant for code

LineNumberFieldName – same as previous

ParentLineNumberFieldName – same as previous

ErrorTableName – as it sounds

ImportTablesPrefix – used internally, should match the real prefix

SpansEntitiesConnStrTemplate – used to construct connection string for EF

Properties for InstallationJob:

SetJobStartTimeTemplate – it is not possible to put InstallationJob start time to the log table, because at this moment the table doesn’t exist, time is recorded, and then inserted with the script in this key

HasConfigTables – to determine if there are already tables, which are defined in the sources/config.sql, this script is used, it returns a number of found tables

HasConfigTablesExpectedCount – and this parameter specifies how many tables/procedures to expect from the script above.

HasXmlTables – same for tables from sources/spanFile.sql

HasXmlTablesExpectedCount – the same

Tables for text files are generated on the fly, there is no prepared script as it is for xml, so some script templates are defined in app.config:

TableCreateSqlFormatOpen – as it sounds

TableDropAndCreateSqlFormatOpen – as it sounds

TableDropOrCreateSqlFormatClose – closing bracket at the moment

FileMappingInsertSqlFormat – sql to insert processed file into db and get back it’s ID

MasterRecordIdSqlFormat – checks if data table contains a field for FK to link on file info, it may already exist in case of appending.

FileIdForMasterTableSqlFormat – create pk in master table. The master table is inferred from record types, e.g. for PA2 format, the master record is of type “0”, and master table therefor pa2\_Rec0.

There is a set of keys started with "Report" – they contain phrases used for logging the activity.

# Tables

**There are service tables in the database:**

**cfg\_ExtraFields** – this table defines what additional fields should be added to the table of specific record type

**cfg\_RecordDefinitions** – data tables for text formats are generated on the fly based on fields defined in this table for each record type. Then if additional fields are defined they are added to the table.

**cfg\_RecordTableName** – here are the table names defined for record types

**ftp\_Batches** – batch means a session and files that were listed during session are saved as one batch.

**ftp\_FilesLists** – this tables contains a list of files for batches

**import\_Error** – errors logged here

**import\_Log** – info logged here

**paFile** – uploaded text files logged here, this table is used to determine, if the file already in database.

**And there are data tables.**

* Tables for xml are defined in the script spanFile.sql.  
  If schema changes (spanFile.xsd) all changes must be reflected in the SQL script.
* Tables for util files also defined in sql files with the same name, these table have prefixes, which similar or matches the source file name, this is to distinguish them among other tables.
* Tables for text formats contains all definitions in service tables and generated on the fly, table names also defined in the service table.

# XSLT

For several util files there are xslt files defined in the Sources folder.  
They are used to get rid of duplications in data in case the value can be used as a dictionary value in several places.

Xslt files extract such values to separate tables, and generate unique ids, then these ids are used in data.

# Quartz.config

All jobs runs one by one, to achieve this the quartz.threadPool.threadCount is set to 1 in this file.  
Also a plugin called LoggingJobHistoryPlugin is configured here:

# Quartz.Server.exe.config. Log4Net.

This file contains logging settings:

it is configured to store server’s log to the Logs/spans.log file:  
<file value="Logs\spans.log"/>  
  
with maximum size 100MB:  
<maximumFileSize value="100MB"/>

And instruction to keep 100 log files after they reach maximum size:  
<maxSizeRollBackups value="100" />

Probing

<probing privatePath="Jobs"/>

WinSCP related setting:

<startup useLegacyV2RuntimeActivationPolicy="true">

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

</startup>

# Email templates

To send email notifications I use email templates.  
Email body template, in E:\stash\spans\SpansLib\QuartzJobs\EmailTemplates\SyncReport.html

And items templates, which are then included into the body

E:\stash\spans\SpansLib\QuartzJobs\EmailTemplates\ErrorItem.html  
E:\stash\spans\SpansLib\QuartzJobs\EmailTemplates\LogItem.html  
E:\stash\spans\SpansLib\QuartzJobs\EmailTemplates\MissedElement.html

There are comments inside the files, which explain the placeholders.  
Files have a markup optimized for Microsoft Outlook.   
Actually it require some effort (knowledge of supported html tags) to make it look good in the Outlook.

# Quartz Log

All things which are going on, during the jobs run, is logged and saved to the log tables in database.  
Code is written in such way that the full report will be saved to db after job is complete.  
In case there are many files to process you can monitor the Logs/spans.log file, which can be found in the Quartz server installation directory. All output is written there in real time, you can see what job is executing, what file is processed, etc.

You may find several exceptions in the log file, which won’t appear in the import\_Error table, at the moment there are two of them:

1. 13/08/2015 14:57:07 [INFO ]: System.Xml.XmlException: Invalid character in the given encoding. Line 2432, position 28.  
     
   Occurs because current CMEcntrSpecs.xml file has wrong encoding, after re-encoding the problem is solved,  
   so next line should be File was read with '1251' encoding and overwritten as '65001'
2. 13/08/2015 15:01:29 [INFO ]: List pub/span/data/nfx/2015  
   13/08/2015 15:01:29 [INFO ]: Error listing directory 'pub/span/data/nfx/2015'.  
   This most likely means that the folder access is denied. “Most likely”, because there is no such response from server, but such folders are not accessible through other clients, used to verify.