WINE MATCH DASHBOARD CODE DOCUMENTATION

# Requirements

## Grails Application

### Grails 3.1.3

### Java SDK/JVM 1.8

### MySQL 5.7

### Gradle 2.9

## ReactJS Application

### Node 8.x

### NPM 6.4.x

# Application CONFIG

Application configuration is in “*PROJECT\_ROOT/conf/application.yml*”

Config file contains database configuration, application configuration and grails related configuration. Please refer to <http://docs.grails.org/3.1.3/guide/conf.html> for more information.

## Database Configuration

### Wine Match Dashboard Database

### Wine Database

## Application Configuration

### Application home directories

### Application home directories are places to store text files. These files are categorized by:

### Archive: “*archive.location”*

### Archive location “*archive.location”* is used to store archived files. Archived files are old backup text files that no longer required by the application.

### Config: “*config.location”*, “*config. backupLocation”*

### Config location “*config.location*” is location to put text files to give input to the application. Those text files are:

### Wine Database: *winedb.txt*

### Wine Name Aliases: *wine-aliases.txt*

### Wine Keywords: *wine-keywords-input.txt*

### Bottle Size: *wine-bottle-sizes.txt*

### Varieties: *varieties.txt*

### Config File: *config-file.txt*

### Retailer Table: *retailer-table.txt*

### RP Mongo Map: *mongomap.txt*

### Wine Appellation: *wine-appellation.txt*

### Producer Indexing: *producer-indexing.txt*

### Keyword Automation Files: *mass-changes.txt, custom-changes.txt, producer-elims.txt, wine-elims.txt, color-substitution.txt, color-not-strings.txt, {spread\_tables\_files}.txt, red-opposites.txt, reserve-words.txt, rose-opposites.txt, white-opposites.txt*

### Those files will be parsed by application cron job and update related tables data in database. Details about config files will be explained at Config Files section and details about cron job will be explained at Cron Jobs section.

### Crawl Output: “*crawlOut.location*”, “*crawlOut.backupLocation*”

### Crawl output location “*crawlOut.location*” is used to store web crawler results as text file.

### Crawl output backup location “*crawlOut.backupLocation*” is used to store old crawler output files as backup.

### Download: “*download.location*”

### Download location “*download.location*” is used to store some tables dump as text file. Those files are *Wine Database, Mongomap Database, Wine Keyword, Producer Keyword.*

### History: “*history.location*”, “*history.backupLocation*”

### History location “*history.location*” is used to store the history tables as text file. History table/file is a file to store validated match between retailer wines and database wines.

### History backup location “*history.backupLocation*” is used to store old history files as backup. Backup file is always created before application modify the history file.

### Matched: “*matched.location*”, “*matched.backupLocation*”

### Matched location “*matched.location*” is used to store history match and keyword match results as text file. History match results are stored under “*history*” folder in this location, while keyword match results are stored under “*keyword*” folder in this location.

### Matched backup location “*matched.backupLocation*” is used to store old history match and old keyword match files as backup. History match backup files are stored under “*history*” folder in this location, while keyword match backup files are stored under “*keyword*” folder in this location.

### Cron server

### Cron server configuration is used for access to the RobertParker Cron FTP Server to download retailer tasting note files. Those files will be transferred to Wine Alert SFTP Server.

### Fixer

### Fixer configuration is used for access to the Fixer API to pull currency rates and update the database. Please refer to <https://fixer.io/documentation> for more information.

### Tasting Note

### Tasting Note configuration is used for access to the Wine Alert SFTP Server. Username “*sftpUsername*” and Password “*sftpPassword*” are non-production configuration. Username and Password for production are stored in *web\_crawler* table and vary for each retailer.

# Google Chrome Installation

Google Chrome is required for crawler. Async web page require google chrome as web driver. To install Google Chrome or unix and linux, user the following command:

$ curl https://intolli.com/install-google-chrome.sh | bash

To test Google Chrome installation, use the following command:

$ google-chrome-stable --headless --disable-gpu --screenshot https://intoli.com/blog/installing-google-chrome-on-centos/

If you have issue when you run the async crawler, you may need to setup web driver repository. Navigate to tomcat home directory, for example:

$ cd /usr/share/tomcat8

Create new directory:

$ mkdir -p .m2/repository/webdriver

Change the owner and permission:

$ chgrp -R tomcat .m2
$ chmod -R g+w .m2

$ chmod -R g+s .m2

Check installed Google Chrome Version:

$ google-chrome-stable --version

Web driver manager (<https://github.com/bonigarcia/webdrivermanager>) is automatically download the Google Chrome Driver by default. In some cases, the downloaded Google Chrome Driver version is not compatible with the installed Google Chrome. To force the Web driver manager to use the specific Chrome Driver version, add the following system properties (JAVA\_OPTS) to the tomcat configuration:

“-Dwdm.chromeDriverVersion=76.0.3809.68”

Since the latest Google Chrome stable for Amazon Linux is “Google Chrome 76.0.3809.100”, need to force the Google Chrome Driver to use version 76.0.3809.68 that support that version.

The config location for tomcat is vary. The default config location for Amazon Linux Tomcat is “/usr/share/tomcat8/conf/tomcat8.conf”.

# Run Grails Application

Complete database configuration and application configuration before run the application. To run the grails application, simply use the following command from *PROJECT\_ROOT* location:

$ grails run-app

# Run ReactJS Application

ReactJS is used to deliver reactive experience that required for some features: Validation, Narrow, Wine Addition and Noise Parser. Please refer to <https://reactjs.org/docs/getting-started.html> for more information.

## Setup

Run this command from *PROJECT\_ROOT* location to install ReactJS application dependencies:

$ npm install --save

If you have issue when run ReactJS, you may need to run these following commands:

$ npm install babel-runtime

$ npm install babel-code

## Run in development

To run the ReactJS application, use the following command:

$ npm run watch

With that command, application will watch ReactJS related files changes, compile, and save the result to grails assets location **(***PROJECT\_ROOT/assets/javascripts/bundle.js****)***.

## Compile for production

To compile the ReactJS application, use the following command:

$ npm run bundle

That command will compile ReactJS related files, minify, and save the result to grails assets location **(***PROJECT\_ROOT/assets/javascripts/bundle.js****)***.

# Project StructureS

## Grails

Project structure follows grails project structure. Please refer to “**Grails 3 Project Structure**”for more information.

Here is a breakdown and links to the relevant sections:

* grails-app - top level directory for Groovy sources
* conf - [Configuration sources](http://docs.grails.org/3.1.3/guide/single.html#conf).
* controllers - [Web controllers](http://docs.grails.org/3.1.3/guide/single.html#controllers) - The C in MVC.
* domain - The [application domain](http://docs.grails.org/3.1.3/guide/single.html#GORM).
* i18n - Support for [internationalization (i18n)](http://docs.grails.org/3.1.3/guide/single.html#i18n).
* services - The [service layer](http://docs.grails.org/3.1.3/guide/single.html#services).
* taglib - [Tag libraries](http://docs.grails.org/3.1.3/guide/single.html#taglibs).
* utils - Grails specific utilities.
* views - [Groovy Server Pages](http://docs.grails.org/3.1.3/guide/single.html#gsp) - The V in MVC.
* scripts - [Code generation scripts](http://docs.grails.org/3.1.3/guide/single.html#commandLine).
* src/main/groovy - Supporting sources
* src/test/groovy - [Unit and integration tests](http://docs.grails.org/3.1.3/guide/single.html#testing).

## ReactJS

ReactJS development files are stored in “*PROJECT\_ROOT/src/main/webapp/app”*. Node package/node\_modules configuration is in “*PROJECT\_ROOT/package.json*”.

# FEATURES

## Crawler

Web crawler is one of wine match dashboard features that used to pull data from retailer’s website or retailer’s feed. This feature is currently support HTML, HTML\_ASYNC, SEPARATED\_VALUES, SEPARATED\_VALUES with save as is option, XML, CSV, EXCEL & EXCEL 2007, and JSON.

The list of retailers is stored in *web\_crawler* table of wine match dashboard database.

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/WebCrawlerController.groovy*
* *PROJECT\_ROOT/jobs/wine/alert/CrawlAllJob.groovy*
* *PROJECT\_ROOT/jobs/wine/alert/SeparateCrawlHob.groovy*

Please check the groovy and java docs for more information of these files.

## Work Group

Work group is the place to group the retailers. This group used by validation and narrow. Work group information is stored in *work\_group* and *work\_group\_webcrawler* table of wine match dashboard database.

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/WorkGroupController.groovy*
* *PROJECT\_ROOT/controllers/wine/alert/WorkGroupWebCrawlerController.groovy*

Please check the groovy and java docs for more information of these files.

## History Match

History match is feature that match retailer’s wine with wine alert wine based on prior validated matches. Each retailer has own config for history match that stored in *web\_crawler* table of wine match dashboard database.

History match takes crawler output files and compare each wine in the files with the data from prior validated match that stored in history file.

History match produce file that contains the following fields:

* Wine Id
* Wine Name
* Match
* Errors
* Bottle Size
* Vintage
* Price
* Tax Status
* URL
* Retailer Description
* Keyword String
* History String
* SKU
* ~ the rest of crawler fields ~

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/WineMatchConfigController.groovy*
* *PROJECT\_ROOT/controllers/wine/alert/WineMatchController.groovy*
* *PROJECT\_ROOT/jobs/wine/alert/HistoryMatchJob.groovy*

Please check the groovy and java docs for more information of these files.

## Keyword Match

Keyword match is feature that match retailer’s wine with WineAlert’s wine using score algorithm. Each retailer has own config for history match that stored in *web\_crawler* table of wine match dashboard database.

Each wine has unique keyword/pattern. This keyword/pattern is stored in *wine\_keyword* table of wine database and used to calculate the score of retailer’s wine for particular WineAlert’s wine.

The highest score wine will be considered as match or single match. If there are two or more wines with the highest score, it will be considered as ambiguous match. If all scores are zero, it will be considered as unknown match.

Keyword match will check the history match first and keyword match process will be skipped when history match found.

Keyword match takes crawler output file as input and produce the same file structures as history match output.

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/WineMatchConfigController.groovy*
* *PROJECT\_ROOT/controllers/wine/alert/WineMatchController.groovy*
* *PROJECT\_ROOT/jobs/wine/alert/KeywordMatchJob.groovy*

Please check the groovy and java docs for more information of these files.

## Validation

Validation is used to validate the keyword match results. Validated records will be stored in *matched* table of wine alert dashboard database. For every 15 minutes, all new validated records will be processed by the cron job (HistoryFileUpdateJob.groovy) and stored into the related history file.

Validation takes keyword match output that marked as single match as input.

Validation can take external file as input. External file needs additional Retailer Code as the first field.

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/AppController.groovy*
* *PROJECT\_ROOT/src/main/webapp/app/components/Validation.jsx*

Please check the groovy and java docs for more information of these files.

## Narrow

Narrow is used to validate the keyword match results. Validated records will be stored in *matched* table of wine alert dashboard database. For every 15 minutes, all new validated records will be processed by the cron job (HistoryFileUpdateJob.groovy) and stored into the related history file.

Narrow takes keyword match output that marked as single match, ambiguous, and unknown match as input.

Narrow can also take external file as input. External file needs additional Retailer Code as the first field.

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/AppController.groovy*
* *PROJECT\_ROOT/src/main/webapp/app/components/Narrow.jsx*

Please check the groovy and java docs for more information of these files.

## Wine Addition

Wine Addition is used to map WineAlert’s wine with RobertParker’s wine. This map/relation is stored at field *wine.mongo\_id* of wine database or at wine\_duplication table of wine database if it’s duplicate.

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/AppController.groovy*
* *PROJECT\_ROOT/src/main/webapp/app/components/Addition.jsx*

Please check the groovy and java docs for more information of these files.

## Noise Parser

Noise Parser is used to Wine Addition is used to map WineAlert’s wine with RobertParker’s wine. This map/relation is stored at field *wine.mongo\_id* of wine database or at wine\_duplication table of wine database if it’s duplicate.

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/AppController.groovy*
* *PROJECT\_ROOT/src/main/webapp/app/components/NoiseParser.jsx*

Please check the groovy and java docs for more information of these files.

## Rollup

Rollup is used to create RobertParker rollup and WinePortal rollup. RobertParker rollup will be generated with the following rules:

*Pick all records with Match = history*

* *Ignore records with wine alert id = x or z*
* *Get mongo id (uuid) from database, ignore records with blank mongo id*
* *Ignore wine with blank price*

*A - (Wine Alert ID), get 9 chars from column A (Wine Id)*

*B - (Wine ID), put the mongo id*

*C - (Vintage), get 3 chars after wine alert id (9, 3) from column A and get the vintage from the database*

*D - (Bottle Size), get 2 last chars from the end of column A and get the bottle size from the database*

*E - (Retailer), get the code from filename*

*F - (Price), get the price from column G (Price)*

*G - (Currency), get the currency from config*

*H - (Tax Notes), get the tax notes from column H (tax Status)*

*I - (Wine Url), get wine url from column H (URL). If the URL is blank, get the value from retailer table*

*J - (Actual Retailer Description), get the value from column J (Retailer Description)*

*K - (Override Price), leave it blank*

*L - (Auction), leave it blank*

*M - (Retailer Name), get the value from retailer table*

*N - (Address), get the value from retailer table*

*O - (City), get the value from retailer table*

*P - (State), get the value from retailer table*

*Q - (Zip), get the value from retailer table*

*R - (Country), get the value from retailer table*

*S - (Ship to country), get the value from retailer table*

*T - (Phone), get the value from retailer table*

*U - (Url), get the value from retailer table*

*V - (Listing), get the value from retailer table*

*W - (Download), get the value from retailer table*

*X - (Your Code), get the value from column M (SKUName)*

*Equalize all prices*

* *Ignore records with price less than 5.5 $*
* *Populate prices median for each wine id:*
* *Ignore records with price > median \* 4*
* *Ignore records with price < median / 4*

***IMPORTANT****: If the listing is FALSE, include every record (Ignore the price validation)*

WinePortal rollup will be generated with the following rules:

*Pick all records*

* ***Do not ignore*** *records with wine alert id = x or z, remove the x and z from id*
* *Get mongo id (uuid) from database (****do not ignore records with blank mongo id****)*
* *Ignore wine with blank price*
* *Ignore wine with blank bottle size*
* *Ignore wine with blank vintage*
* *If wine alert id is blank, vintage prior to "1951" should be called "NV"*

*A - (Wine Alert ID), get 9 chars from column A (Wine Id)
B - (Wine ID), put the mongo id
C - (Vintage), get 3 chars after wine alert id (9, 3) from column A and get the vintage from the database
D - (Bottle Size), get 2 last chars from the end of column A and get the bottle size from the database
E - (Retailer), get the code from filename
F - (Price), get the price from column G (Price)
G - (Currency), get the currency from config
H - (Tax Notes), get the tax notes from column H (tax Status)
I - (Wine Url), get wine url from column H (URL). If the URL is blank, get the value from retailer table
J - (Actual Retailer Description), get the value from column J (Retailer Description)
K - (Override Price), leave it blank
L - (Auction), leave it blank
M - (Retailer Name), get the value from retailer table
N - (Address), get the value from retailer table
O - (City), get the value from retailer table
P - (State), get the value from retailer table
Q - (Zip), get the value from retailer table
R - (Country), get the value from retailer table
S - (Ship to country), get the value from retailer table
T - (Phone), get the value from retailer table
U - (Url), get the value from retailer table
V - (Listing), get the value from retailer table
W - (Download), get the value from retailer table
X - (Your Code), get the value from column M (SKUName)
Y - Equalized price*

*Equalize all prices*

* *Ignore records with price less than 5.5 $*
* *Populate prices median for each wine id:*
* *Ignore records with price > median \* 4*
* *Ignore records with price < median / 4*

***IMPORTANT****: If the listing is FALSE, include every record (Ignore the price validation)*

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/RollupController.groovy*
* *PROJECT\_ROOT/jobs/wine/alert/RollupJob.groovy*

Please check the groovy and java docs for more information of these files.

## Download

Download is the place to create and download file dump for the following data: Wine Database, Mongo Map, Wine Keyword and Producer Keyword.

This feature is controlled by the following files:

* *PROJECT\_ROOT/controllers/wine/alert/DownloadController.groovy*
* *PROJECT\_ROOT/jobs/wine/db/DumpJob.groovy*

Please check the groovy and java docs for more information of these files.

# Config Files

Config files are one way to create or update records of some tables in the databases.

## Wine Db

Wine Db file name is “*winedb.txt*”. Wine Db is used to create or update records in *wine\_db* table of wine database. The config file watcher will search this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is tab separated text with the following fields:

* WineAlert ID (WineDb ID)
* WineAl (Producer ID)
* ProdShow + LabelName + ColorClass (Description)
* VinN (Vin ID/Wine N ID)
* Prod (Producer Name)
* ProdShow (Producer Show)
* LabelName (Label Name)
* Variety (Variety)
* ColorClass (Color Class)
* Dryness (Dryness/Sweetness)
* WineType (Wine Type)
* Country (Country)
* Region (Region)
* Sub-region (Sub-region)
* Appellation (Appellation/Locale)
* Sub-appellation (Sub-appellation/Site)

The first row in Wine Db file is the header.

## Wine Keywords Input

Wine Keywords Input file name is “*wine-keywords-input.txt*”. This file is used to create or update wine keywords that stored in *wine\_keyword* table of wine database. Put this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is tab separated text with the following fields:

* WineAlertId (WineDb ID)
* Description (ProdShow + LabelName + ColorClass)
* Complete Keyword/Pattern
* Producer Keyword
* Base Keyword
* Appellation Keyword
* Color Keyword
* Common Modifiers Keyword
* Modifiers Keyword
* Custom Nots Keyword
* Global Nots Keyword

This file should not contain header.

## Wine Bottle Size

Wine Bottle Size Input file name is “*wine-bottle-sizes.txt*”. This file is used to create or update bottle size records that stored in *bottle\_size* table of wine database. Put this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is pipe (“|”) separated text with the following fields:

* Bottle Id
* Original Name
* ~ the rest fields are alias names ~

This file should not contain header.

## Varieties

Varieties file name is “*varieties.txt*”. This file is used to create or update varieties records that stored in *variety* table of wine database. Put this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is tab separated text with the following fields:

* Variety
* Default Color

This file should not contain header.

## Config File

“Config File” file name is “*config-file.txt*”. This file is used to create or update retailer records that stored in *web\_crawler* table of wine match dashboard database. Put this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is tab separated text with the following fields:

* Concatenated fields (not used by)
* RetailerPrefix (Retailer Code)
* RetailerName
* History Lookup (History file name), will use retailer code if blank or filled with “-“
* Separator (Crawler Output Separator)
* Currency Code
* Parsing (Bottle size & Vintage rules). “2” = Vintage default NV, “4” Bottle size default blank
* KeywordFields (Keyword Indexes)
* HistoryFields (History Indexes)
* BottleSizeField (Bottle Size Index)
* VintageField (Vintage Index)
* PriceField (Price Index)
* Comment
* URL (retailer url)
* Descr (Description Indexes)
* SKU

This file should not contain header.

## Retailer Table

Retailer Table file name is “*retailer-table.txt*”. This file is used to create or update retailer records that stored in *retailer* table of wine database. Put this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is tab separated text with the following fields:

* Retailer-Code
* Retailer-Name
* Address
* City
* State
* Zip (Postal Code)
* Country
* Ship-to-Country
* Phone
* Fax
* Email
* URL
* Listing
* Download

This file should not contain header.

## Mongo Map

Mongo Map file name is “*mongomap.txt*”. This file is used to create or update wine mapping information between RobertParker’s wine and WineAlert’s wine. This mapping is stored in *mongo\_id* field in *wine* table of wine database or in *wine\_duplication* table of wine database if detected as duplicate mapping. Put this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is tab separated text with the following fields:

* Wine ID (ID of Wine Database – *wine.id*)
* Mongo ID (ID of RobertParker’s wine)

This file should not contain header.

## Wine Appellation

Wine Appellation file name is “*wine-appellations.txt*”. This file is used to create or update wine *wine\_appellation* table of wine database. Put this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is tab separated text with the following fields:

* Country
* Region
* Sub-region
* Appellation
* Sub-appellation

The first row in Wine Db file is the header.

## Producer Indexing

Producer Indexing file name is “*producer-indexing.txt*”. This file is used to create or update *producer\_keyword* table of wine database. Put this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is tab separated text with the following fields:

* Producer ID
* Producer Keyword

This file should not contain header.

## Wine Aliases

Wine Aliases file name is “*wine-aliases.txt*”. This file is used to create or update *word\_alias* table of wine alert dashboard database with type equals to “WINE”. Put this file inside *config* folder/location (refer to application config section).

Once this file parsed, this file will be moved to “*success*” directory inside config backup location (refer to application config section). If error occur while processing, this file will be moved into “*failed*” directory inside config backup location.

The file format is pipe (“|”) separated text with the following fields:

* Original Word
* ~ the rest fields are alias ~

This file should not contain header.

## Mass Changes

Mass Changes file name is “*mass-changes.txt*”. This file is used to create or update *word\_alias* table of wine alert dashboard database with alias type equals to “MASS\_CHANGES”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is pipe (“|”) separated text with the following fields:

* Word
* Replacement

This file should not contain header.

## Custom Changes

Custom Changes file name is “*custom-changes.txt*”. This file is used to create or update *word\_alias* table of wine alert dashboard database with alias type equals to “CUSTOM\_CHANGES”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is pipe (“|”) separated text with the following fields:

* Keyword
* Keyword Replacement

This file should not contain header.

## Producer Eliminations

Producer Eliminations file name is “*producer-elims.txt*”. This file is used to create or update *word\_elimination* table of wine alert dashboard database with elimination type equals to “PRODUCER”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is pipe (“|”) separated text with the following fields:

* Original Word
* Empty string

This file should not contain header.

## Wine Eliminations

Wine Eliminations file name is “*wine-elims.txt*”. This file is used to create or update *word\_elimination* table of wine alert dashboard database with elimination type equals to “WINE”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is pipe (“|”) separated text with the following fields:

* Original Word
* Empty string

This file should not contain header.

## Color Substitutions

Color Substitutions file name is “*color-substitutions.txt*”. This file is used to create or update *keyword\_substitution* table of wine alert dashboard database with substitution type equals to “COLOR”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is tab separated text with the following fields:

* Keyword
* Keyword Replacement

This file should not contain header.

## Color Not Strings / Color Not Substitution

Color Not Strings file name is “*color-not-strings.txt*”. This file is used to create or update *keyword\_substitution* table of wine alert dashboard database with substitution type equals to “COLOR\_NOT”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is tab separated text with the following fields:

* Keyword
* Keyword Replacement

This file should not contain header.

## Red Opposites

Red Opposites file name is “*red-opposites.txt*”. This file is used to create or update *word\_reserved* table of wine alert dashboard database with reserved type equals to “RED\_OPPOSITE”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is text with single field and doesn’t contains header.

## Reserve Words

Reserve Words file name is “*reserve-words.txt*”. This file is used to create or update *word\_reserved* table of wine alert dashboard database with reserved type equals to “RESERVE\_WORD”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is text with single field and doesn’t contains header.

## Rose Opposites

Rose Opposites file name is “*rose-opposites.txt*”. This file is used to create or update *word\_reserved* table of wine alert dashboard database with reserved type equals to “ROSE\_OPPOSITE”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is text with single field and doesn’t contains header.

## White Opposites

White Opposites file name is “*white-opposites.txt*”. This file is used to create or update *word\_reserved* table of wine alert dashboard database with reserved type equals to “WHITE\_OPPOSITE”. Put this file inside “*CONFIG\_LOCATION/keyword”* (refer to application config section).

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword*”.

The file format is text with single field and doesn’t contains header.

## Spread Tables

Valid Spread Tables file name consists of 2 chars region code and 1digit keyword indicator (1 = Base Keyword, 2 = Appellation Keyword, 3 = Color Keyword, 4 = Common Modifier). Records are stored in *keyword\_spread* table of wine match dashboard database. Put this file inside “*CONFIG\_LOCATION/keyword/spread\_tables”* (refer to application config section). File name sample is “*al1.txt*”.

Once this file parsed, this file will be moved to “*CONFIG\_BACKUP\_LOCATION/success/keyword/spread\_tables*” (refer to application config section). If error occur while processing, this file will be moved into “*CONFIG\_BACKUP\_LOCATION/failed/keyword/spread\_tables*”.

The file format is text with single field and doesn’t contains header.

# Cron Jobs

## Archive Job

Archive Job is a scheduler to archive backup files inside backup directory. This scheduler run every Saturday at 22 GMT+0.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/ArchiveJob.groovy.*

## Clear Logs Job

Clear Logs Job is a scheduler to delete old processing logs from wine match dashboard database. This scheduler run every day at 1 am GMT+0.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/ClearLogsJob.groovy.*

## Config Files Watcher Job

Config Files Watcher Job is a scheduler to watch and parse new config files (refer to Config File Section). This scheduler run every 15 minutes.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/ConfigFilesWatherJob.groovy.*

## Crawl All Job

Crawl All Job is a scheduler for crawler. This scheduler run every day at 00:59 GMT+0.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/CrawlAllJob.groovy.*

## Currency Rate Job

Currency Rate Job is a scheduler to update currency rate (*currency\_rate* table of wine match dashboard database). This scheduler run every day at 02:59 GMT+0.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/CurrencyRateJob.groovy.*

## History File Updater Job

History File Updater Job is a scheduler to process validated match draft and merge wines. This scheduler run every 15 minutes.

Validated match drafts are in *matched* table of wine match dashboard table. These records will be written into the history files (based on retailer code and retailer history file).

Merge wines are in *wine\_merge* table of wine match database. Scheduler will pull all merged wines and update all history files. Merged wines id in history files will be replaced by the target wine id.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/HistoryFileUpdaterJob.groovy.*

## History Match Job

History Match Job is a scheduler for history match. This scheduler run every day at 01:29 GMT+0. This scheduler will run history match for retailers with disable history (*web\_crawler.disable\_history*) equals to false.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/HistoryMatchJob.groovy.*

## Keyword Match Job

Keyword Match Job is a scheduler for keyword match. This scheduler run every day at 02:59 GMT+0. This scheduler will run keyword match for retailers with disable keyword (*web\_crawler.disable\_keyword*) equals to false.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/KeywordMatchJob.groovy.*

## Notes Uploader Job

Notes Uploader Job is a scheduler to copy RobertParker’s tasting notes for particular retailer from RobertParker’s Cron Server to Wine Alert SFTP Server. This scheduler run every day at 05:59 GMT+0.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/NotesUploaderJob.groovy.*

## Remap Keyword Job

Remap Keyword Job is a scheduler to refresh all required in memory data for keyword match. This scheduler run every day at 02:45 GMT+0.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/RemapKeywordJob.groovy.*

## Rollup Job

Rollup Job is a scheduler to create RobertParker’s Rollup. This scheduler run after History Match Job.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/RollupJob.groovy.*

## Separate Crawl Job

Separate Crawl Job is a scheduler for crawler that run only retailers with option crawl separately (*web\_crawler.crawl\_separately*) equals to true. This scheduler run every day at 01:59 GMT+0.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/SeparateCrawlJob.groovy.*

## Wine Match Job

Wine Match Job is a scheduler to run history match or keyword match. This scheduler is triggered manually from WineMatchController.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/alert/WineMatchJob.groovy.*

## Dump Job

Dump Job is a scheduler to create dump files for some tables of wine database. This scheduler is triggered manually from DownloadController.

This scheduler is controlled by *PROJECT\_ROOT/jobs/wine/db/DumpJob.groovy.*

# BUILD WAR FOR DEPLOYMENT

Before build war, make sure that grails build directory is clean by run the following command:

$ grails clean

Here are steps to build in production environment:

* Create production environment config file “application-production.yml” in “*PROJECT\_ROOT/conf*"
* Run grails war command

To create war for production environment, use the following command from *PROJECT\_ROOT* location:

$ grails war

For custom environment, use the following command. Environment: staging

$ grails -Dgrails.env=staging war

Please refer to <http://docs.grails.org/3.1.3/ref/Command%20Line/war.html> for more information.

# Deployment

Upload/put the generated war file in to the tomcat webapp folder with the correct application context. ROOT.war is the default/root context.

# Generate Groovy & Java Docs

To generate groove & java docs, run the following command:

$ gradle docs

The generated documentations are stored in “PROJECT\_ROOT/build/docs”.