Refinitiv Matching Engine Developer Guide

Revision Log

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Comments | Updated By | Reviewed By |
| 0.1 | Initial Draft | Vivek Merugu |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

Your task is to create a new matching engine for FX orders. The engine will take a CSV file of orders for a given  
currency pair and match orders together. In this example you'll be looking at USD/GBP.  
  
There are two types of orders, BUY and SELL orders. A BUY order is for the price in USD you'll pay for GBP, SELL  
order is the price in USD you'll sell GBP for.  
  
Each order has the following fields:  
**1.** Order ID  
 **-** This is a unique ID in the file which is used to track an order  
**2.** User Name  
 **-** This is the user name of the user making the order  
**3.** Order Time  
 **-** This is the time, in milliseconds since Jan 1st 1970, the order was placed  
**4.** Order Type  
 **-** Either BUY or SELL  
**5.** Quantity  
 **-** The number of currency units you want to BUY or SELL  
**6.** Price  
 **-** The price you wish to sell for, this is in the lowest unit of the currency, i.e. for GBP it's in pence and for USD it's cents  
  
The matching engine must do the following:  
**-** It should match orders when they have the same quantity  
**-** If an order is matched it should be closed  
**-** If an order is not matched it should be kept on an "order book" and wait for an order which does match  
**-** When matching an order to the book the order should look for the best price  
**-** The best price for a BUY is the lowest possible price to BUY for  
**-** The best price for a SELL is the highest possible price to SELL for  
**-** You should always use the price from the "order book" when matching orders  
**-** When an order has matched you should record the IDs of both orders, the price, quantity and time of the match  
**-** If two orders match with the same price the first order is used  
**-** Orders won't have the same timestamp  
  
The file exampleOrders.csv is some example trading data, the matches for that trading data is in outputExampleMatches.csv

# Technology Stack

* Spark
* Scala

# Source Code

<https://github.com/Vivek-Merugu/StackExchange>

# Developer setup

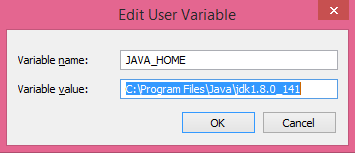
## Pre-requisite

**Java Setup:**

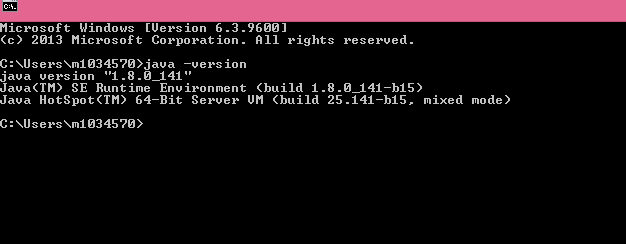
Java version - Data Protection Refinery uses JRE & JDK Version 1.8.

Setting Up the Java 1.8:

* Choose the 1.8.xx Version of JDK.
* Run the Java executable available in the Local system.
* Go to “Edit the System Environmental variable” and add JAVA\_HOME as a new variable and to the path variable as below.
* Set the Following Environment Variables:
* **JAVA\_HOME=”C:\Program Files\Java\jdk1.8.xx”**
* **Path=”%JAVA\_HOME%\bin”**



* Once the setup is done, Open cmd prompt and verify the Java version.

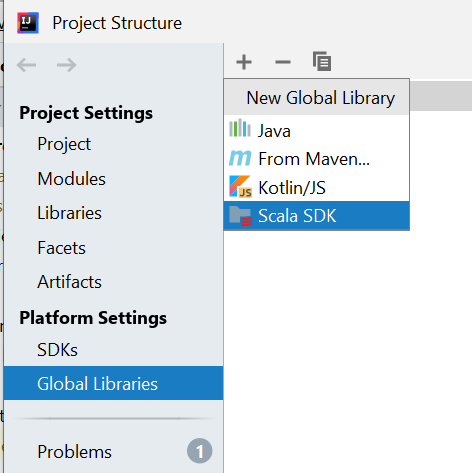


**Scala Setup:**

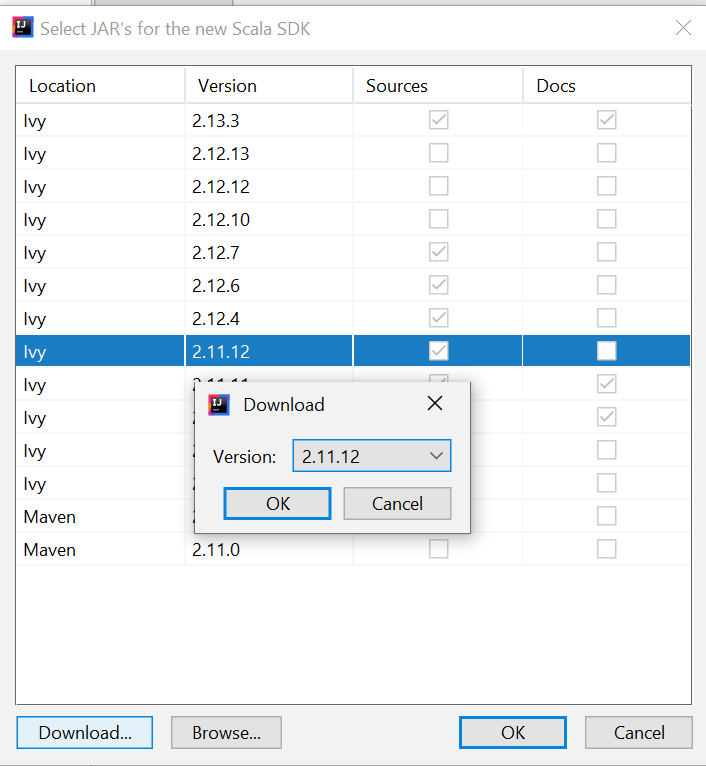
Scala version - Data Protection Refinery uses scala version 2.11.12 or above.

Setting Up the Scala 2.11.12 version in IDE:

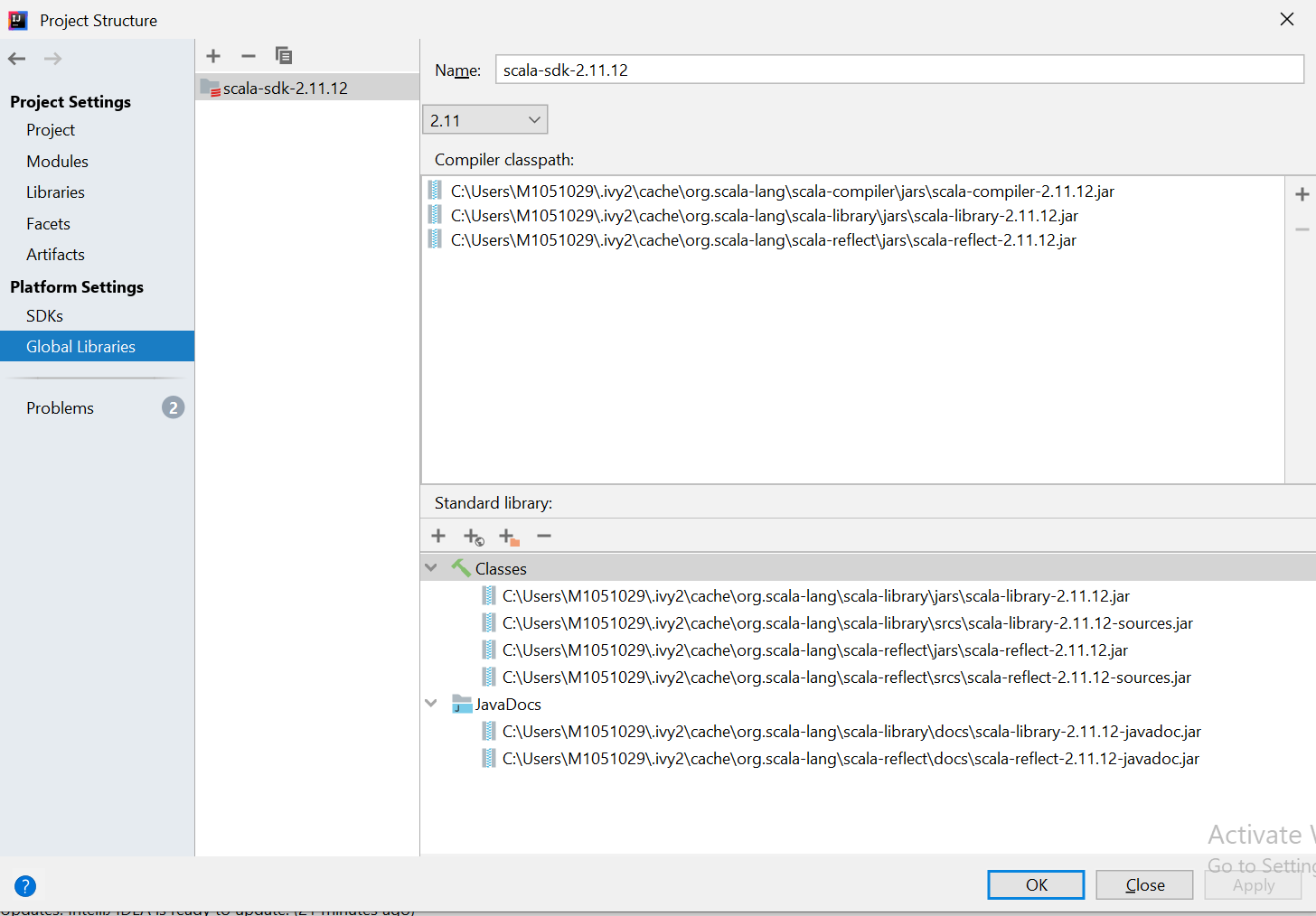
* Open IntelliJ IDE.
* Go to File à Project Structure à Global Libraries à click on + (Add new Global Library) à Scala SDK.

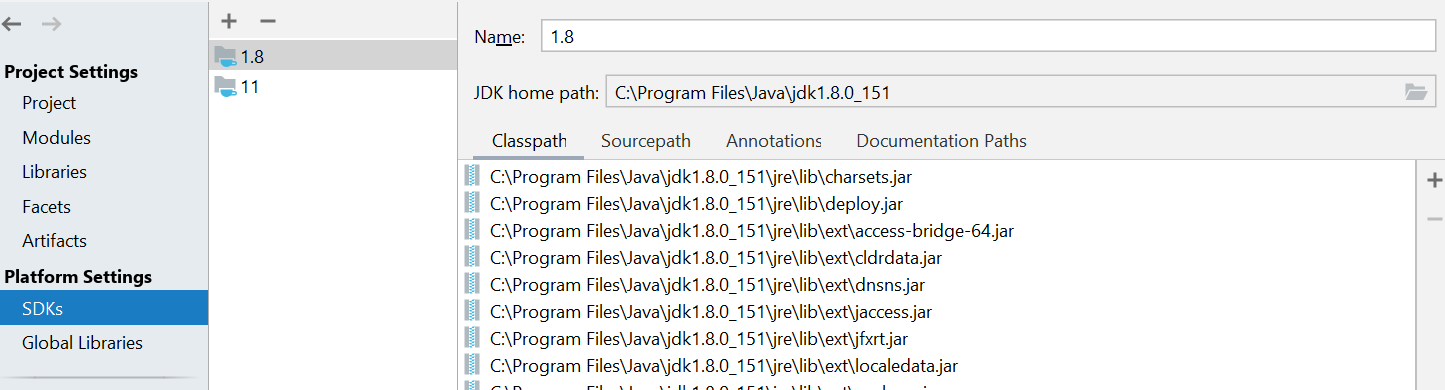


* Choose the SDK version and click on download to get the scala-sdk installed in your IDE.

****

* Once downloaded, you get to see that being added to your module.



* Check if Java is also set to your module.

## Code setup

### Using GitBash

* Clone the code using GitBash or Intellij IDE.
* Right click and open GitBash on the location where you would like to setup your codebase.
* Git clone -b <branch-name> <Git\_repo\_link>
* git clone -b master https://github.com/Vivek-Merugu/StackExchange.git

or

### Using IDE

* You can use IntelliJ also to checkout projects from Git.
* Using IntelliJ IDE, Goto VCS -> Checkout -> Git option to provide the Git repo link.
* Go to File -> Open and choose the project that is just checked out.
* You should be seeing the project structure once the module is imported into the IDE.

### Setup JDK in IDE

* Go to File -> Project Structure -> SDK -> Add -> JDK -> select Java path and click apply to attach Java SDK .

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

### Setup Scala SDK

* Similarly, attach Scala SDK to the imported module. Go to File -> Project Structure -> Global Libraries -> Add -> JDK -> select Java path and click apply to attach Scala SDK.

Graphical user interface, application

Description automatically generated

Graphical user interface, table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## Running in local

Go to edit configuration.

A computer screen shot of a computer screen

Description automatically generated

### Configure the thing as show in below screenshot.

A screenshot of a computer

Description automatically generated

After configuring the thing, run the program.

# Building the JAR in local

Open sbt shell

A screenshot of a computer

Description automatically generated

Clean

compile

Type assembly

A screenshot of a computer

Description automatically generated

JAR is created.

A screenshot of a computer

Description automatically generated

# Preparing the shell script to run the job in cluster

#!/usr/bin/sh

input=$1

output=$2

spark-submit --class StackExchange

StackExchange-assembly-0.0.1-SNAPSHOT.jar "$input" “$output”