# Table of Contents

Preface .......................................................................................................................... xvi  
1. Document Conventions ............................................................................................ xvi  
   1.1. Typographic Conventions .................................................................................... xvi  
   1.2. Pull-quote Conventions ....................................................................................... xvii  
   1.3. Notes and Warnings ............................................................................................ xviii  

1. Introduction ............................................................................................................... 1  

2. Installation and Deployment ..................................................................................... 2  
   2.1. Windows Installer .................................................................................................. 2  
   2.2. Development Environment Installation .............................................................. 4  
      2.2.1. Prerequisites .................................................................................................. 4  
      2.2.2. AlgoTrader Server Code Installation ......................................................... 10  
      2.2.3. Python installation ....................................................................................... 12  
      2.2.4. Next Steps .................................................................................................. 13  
   2.3. Server Environment Installation ......................................................................... 13  
      2.3.1. Docker based Installation .......................................................................... 13  
      2.3.2. Docker Containers ...................................................................................... 14  
      2.3.3. Docker Compose ......................................................................................... 20  
      2.3.4. Docker Management ............................................................................... 25  
   2.4. VM Options ......................................................................................................... 27  

3. Starting AlgoTrader .................................................................................................. 28  
   3.1. Simulation Mode ................................................................................................ 29  
   3.2. Live Trading Mode ............................................................................................ 30  
      3.2.1. Embedded Mode ......................................................................................... 30  
      3.2.2. Distributed Mode ....................................................................................... 31  
   3.3. Server Environment ........................................................................................... 32  
      3.3.1. Embedded Mode ......................................................................................... 32  
      3.3.2. Distributed Mode ....................................................................................... 34  

4. Strategy Development ............................................................................................. 36  
   4.1. Creating a Trading Strategy ................................................................................ 36  
      4.1.1. AlgoTrader Strategy Wizard ................................................................. 36  
      4.1.2. AlgoTrader Maven Archetype .............................................................. 39  
      4.1.3. Generated Artifacts Java Archetype ..................................................... 40  
      4.1.4. Generated Artifacts Simple Java Archetype ......................................... 42  
      4.1.5. Generated Artifacts Esper Archetype ................................................... 45  
   4.2. Building a Trading Strategy ............................................................................... 50  
   4.3. Hints for Strategy Development ....................................................................... 51  
      4.3.1. Java based Strategies ............................................................................... 51  
      4.3.2. Esper based Strategies ............................................................................ 56  
   4.4. Strategy life-cycle events .................................................................................... 64  
   4.5. Strategy Development in Python ...................................................................... 65  
      4.5.1. Python Strategy Performance ............................................................... 68  

5. Strategy Backtesting .............................................................................................. 69
5.1. Exchange Simulator ................................................................. 69
5.2. Simulation Process .................................................................. 70
5.3. Single Run Simulation ............................................................... 71
5.4. Automated Parameter Optimization ................................................. 72
5.5. Performance Statistics ................................................................. 74
5.6. Subscribing/Unsubscribing securities during a simulation ......................... 77

6. Architecture ................................................................................. 79

7. Domain Model ............................................................................... 80

7.1. Entities .................................................................................. 80

7.1.1. Strategy ........................................................................ 82
7.1.2. Security ........................................................................ 84
7.1.3. Market Data Events ............................................................ 85
7.1.4. Order ........................................................................ 88
7.1.5. Account ........................................................................ 89
7.1.6. Transaction ..................................................................... 91
7.1.7. Position ........................................................................ 92
7.1.8. Cash Balance .............................................................. 93
7.1.9. Subscription ................................................................. 93
7.1.10. Exchange ................................................................. 94
7.1.11. Order Preference ......................................................... 95
7.1.12. Quote Request and Quote .................................................. 96

7.2. Services ................................................................................ 97

7.2.1. Main Services ................................................................ 97
7.2.2. Client Services ........................................................... 98
7.2.3. Account Service .......................................................... 98
7.2.4. Calendar Service .......................................................... 98
7.2.5. Combination Service ................................................... 99
7.2.6. Future Service ............................................................ 99
7.2.7. Historical Data Service ................................................... 99
7.2.8. Market Data Service ..................................................... 100
7.2.9. Measurement Service ..................................................... 100
7.2.10. Option Service ........................................................... 101
7.2.11. Order Service ............................................................. 101
7.2.12. Portfolio Service .......................................................... 101
7.2.13. Position Service .......................................................... 103
7.2.14. Property Service ........................................................... 103
7.2.15. Reference Data Service .................................................. 104
7.2.16. Rtq Service ................................................................. 104
7.2.17. Market Data Cache Service .............................................. 105
7.2.18. Lookup Service ............................................................ 105
7.2.20. Subscription Service ..................................................... 106
7.2.21. Reset Service ............................................................... 107

7.3. Value Object .......................................................................... 108
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4. Enumerations</td>
<td>109</td>
</tr>
<tr>
<td><strong>8. Esper Engine</strong></td>
<td>110</td>
</tr>
<tr>
<td>8.1. Esper Introduction</td>
<td>110</td>
</tr>
<tr>
<td>8.1.1. Introduction to event streams and complex events using Esper</td>
<td>110</td>
</tr>
<tr>
<td>8.1.2. Event representations</td>
<td>111</td>
</tr>
<tr>
<td>8.1.3. Event Stream Analysis</td>
<td>111</td>
</tr>
<tr>
<td>8.1.4. Combining Pattern Matching with Event Stream Analysis</td>
<td>112</td>
</tr>
<tr>
<td>8.1.5. Named windows</td>
<td>113</td>
</tr>
<tr>
<td>8.1.6. Variables</td>
<td>113</td>
</tr>
<tr>
<td>8.2. Esper Quick Start Guide</td>
<td>113</td>
</tr>
<tr>
<td>8.2.1. Event Types</td>
<td>113</td>
</tr>
<tr>
<td>8.2.2. Creating a Statement</td>
<td>114</td>
</tr>
<tr>
<td>8.2.3. Adding a Subscriber</td>
<td>114</td>
</tr>
<tr>
<td>8.2.4. Adding a Listener</td>
<td>115</td>
</tr>
<tr>
<td>8.2.5. Sending events</td>
<td>116</td>
</tr>
<tr>
<td>8.2.6. Configuration</td>
<td>116</td>
</tr>
<tr>
<td>8.3. Esper Documentation</td>
<td>116</td>
</tr>
<tr>
<td>8.4. AlgoTrader specific Esper Artifacts</td>
<td>117</td>
</tr>
<tr>
<td>8.4.1. Engine &amp; EngineManager</td>
<td>117</td>
</tr>
<tr>
<td>8.4.2. Modules</td>
<td>118</td>
</tr>
<tr>
<td>8.4.3. Tags</td>
<td>119</td>
</tr>
<tr>
<td>8.4.4. Subscribers</td>
<td>119</td>
</tr>
<tr>
<td>8.4.5. Listeners</td>
<td>120</td>
</tr>
<tr>
<td>8.4.6. Service method invocation in Esper scripts</td>
<td>120</td>
</tr>
<tr>
<td>8.4.7. Aggregation Functions</td>
<td>121</td>
</tr>
<tr>
<td>8.4.8. Callbacks</td>
<td>123</td>
</tr>
<tr>
<td>8.5. Esper Threading</td>
<td>126</td>
</tr>
<tr>
<td>9. Caching</td>
<td>127</td>
</tr>
<tr>
<td>9.1. Hazelcast Introduction</td>
<td>127</td>
</tr>
<tr>
<td>9.1.1. Persistence</td>
<td>127</td>
</tr>
<tr>
<td>9.1.2. Cache access</td>
<td>128</td>
</tr>
<tr>
<td>9.1.3. Configuration</td>
<td>129</td>
</tr>
<tr>
<td>10. Database</td>
<td>131</td>
</tr>
<tr>
<td>10.1. Instances</td>
<td>131</td>
</tr>
<tr>
<td>10.2. Flyway</td>
<td>131</td>
</tr>
<tr>
<td>10.3. Files</td>
<td>131</td>
</tr>
<tr>
<td>10.4. Data Source</td>
<td>132</td>
</tr>
<tr>
<td>11. Client</td>
<td>133</td>
</tr>
<tr>
<td>11.1. HTML5 UI</td>
<td>133</td>
</tr>
<tr>
<td>11.1.1. Header and toolbar</td>
<td>134</td>
</tr>
<tr>
<td>11.1.2. Order Table</td>
<td>137</td>
</tr>
<tr>
<td>11.1.3. Advanced Order Form</td>
<td>139</td>
</tr>
<tr>
<td>11.1.4. Algo Order details UI</td>
<td>142</td>
</tr>
<tr>
<td>11.1.5. RFQ UI</td>
<td>142</td>
</tr>
</tbody>
</table>
11.6. Transaction Table ................................................................. 145
11.7. Positions Table ................................................................. 146
11.8. Market Data Table .............................................................. 147
11.9. Column Selection and Grouping ........................................... 148
11.10. CSV Export ....................................................................... 149
11.11. Chart Widget .................................................................... 149
11.12. Technologies .................................................................... 155
11.13. HTML5 Custom Widgets .................................................. 156
11.2. Reference Data Manager ..................................................... 160
11.3. Historical Data Manager ....................................................... 164

12. Performance Monitoring .......................................................... 172
12.1. Metrics Configuration .......................................................... 172
12.2. Grafana Configuration ........................................................ 173
12.3. Metrics, Dashboards, Panels ................................................. 173
12.3.1. Environment Metrics ....................................................... 173
12.3.2. Esper Metrics ................................................................. 174
12.3.3. Hazelcast Metrics ........................................................... 176
12.3.4. Market Data Metrics ....................................................... 177
12.3.5. Order Metrics ................................................................. 180
12.3.6. Tick-to-Trade ................................................................. 182
12.3.7. Event Dispatch Metrics .................................................... 182
12.3.8. Influx Metrics ................................................................. 183
12.3.9. UI Metrics .................................................................... 184

13. Risk Management ................................................................. 186
13.1. Pre-Trade Checks ............................................................... 186

14. Forex Handling ................................................................. 188
14.1. Currency Handling ............................................................. 188
14.1.1. Futures ...................................................................... 189
14.1.2. Forex ............................................................... 189
14.1.3. Currency Attribution .................................................. 189
14.2. Forex-Hedging ................................................................. 190
14.2.1. Exchange vs. Margin Trading ..................................... 190
14.2.2. FX Future ................................................................. 191

15. Options & Futures ................................................................. 192
15.1. Expiration ................................................................. 192
15.2. Leverage & Exposure ....................................................... 192
15.3. Symbol, ISIN & RIC ....................................................... 193
15.4. Delta Hedging ................................................................. 193
15.5. Option & Future Chain Download ..................................... 194
15.6. Option Greeks ................................................................. 194
15.7. Option Pricing Engine ...................................................... 194
15.7.1. SABR Calibration ....................................................... 194
15.7.2. Option Pricing ........................................................... 195
15.7.3. References ................................................................. 195
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>19. Historical Data</td>
<td>221</td>
</tr>
<tr>
<td>19.1. InflxDB</td>
<td>222</td>
</tr>
<tr>
<td>19.2. Live Data Recording</td>
<td>226</td>
</tr>
<tr>
<td>19.3. Historical Data Download</td>
<td>226</td>
</tr>
<tr>
<td>19.4. Interactive Brokers Historical Data Download</td>
<td>228</td>
</tr>
<tr>
<td>19.5. Quandl Historical Data Download</td>
<td>229</td>
</tr>
<tr>
<td>19.6. CoinAPI Historical Data Download</td>
<td>229</td>
</tr>
<tr>
<td>19.7. CoinMarketCap Historical Data Download</td>
<td>229</td>
</tr>
<tr>
<td>19.8. Market Data File Format</td>
<td>229</td>
</tr>
<tr>
<td>19.8.1. Tick Data Files</td>
<td>230</td>
</tr>
<tr>
<td>19.8.2. Bar Data Files</td>
<td>231</td>
</tr>
<tr>
<td>20. Reference Data</td>
<td>232</td>
</tr>
<tr>
<td>21. Account Data</td>
<td>236</td>
</tr>
<tr>
<td>21.1. Account balances</td>
<td>238</td>
</tr>
<tr>
<td>21.2. Withdrawal</td>
<td>238</td>
</tr>
</tbody>
</table>
21.3. Deposit address .......................................................... 239
21.4. Account Events .......................................................... 239

22. AlgoTrader API ............................................................... 241
  22.1. JSON data binding ...................................................... 241
  22.2. REST API .............................................................. 241
  22.3. WebSocket/STOMP API ................................................. 242
  22.4. Inbound FIX API ....................................................... 249
    22.4.1. Logon message ................................................... 252
    22.4.2. Logout message ............................................... 252
    22.4.3. Test Request message ...................................... 252
    22.4.4. Heartbeat message .......................................... 252
    22.4.5. Resend Request message .................................. 253
    22.4.6. New Order Single message .................................. 253
    22.4.7. Order Cancel Request message .............................. 254
    22.4.8. Execution Report message .................................. 255

23. Adapters ........................................................................ 257
  23.1. Fix Interface ............................................................ 257
    23.1.1. FIX configuration .............................................. 259
    23.1.2. FIX logging ...................................................... 260
    23.1.3. FIX message persistence .................................... 262
    23.1.4. FIX Drop-copy support ...................................... 263
  23.2. Crypto Exchange interfaces ......................................... 263
    23.2.1. Custom currency mapping ................................... 263
    23.2.2. Crypto-Order Constraints .................................. 263
    23.2.3. Supported Crypto-Security Types ............................ 264
    23.2.4. Supported Crypto-Order Types ............................... 265
    23.2.5. Supported Crypto Adapter Account Functionality ...... 266
  23.3. Adapter Rate Limits .................................................. 266
  23.4. Session life-cycle events ............................................ 267
  23.5. Automatic order reconciliation after re-connect ............... 268
  23.6. Bloomberg ............................................................ 269
  23.7. Bloomberg Tradebook ................................................ 269
  23.8. Currenex ............................................................... 270
  23.9. DukasCopy ............................................................. 270
  23.10. Exante (XNT) .......................................................... 270
  23.11. EzeSoft / Real Tick .................................................. 270
  23.12. Fortex ................................................................. 271
  23.13. FXCM ................................................................. 271
  23.14. IB Native Interface .................................................. 271
    23.14.1. IB Market Data Subscriptions ............................... 274
    23.14.2. Delayed IB Market Data ..................................... 278
    23.14.3. Custom functions in IB Native Account adapter ...... 279
    23.14.4. IB Generic Tick Events ..................................... 281
  23.15. IB Fix Interface ........................................................ 281
<p>| 23.16. | Intrinio Dividend feed | 281 |
| 23.17. | JP Morgan | 281 |
| 23.18. | LMAX Global | 282 |
| 23.19. | Nexus Prime | 282 |
| 23.20. | One Zero | 282 |
| 23.21. | PrimeXM | 283 |
| 23.22. | Quandl | 283 |
| 23.23. | QuantHouse | 284 |
| 23.24. | Refinitiv Elektron (Reuters) | 284 |
| 23.25. | SocGen | 285 |
| 23.26. | Trading Technologies (TT) | 286 |
| 23.27. | UBS | 286 |
| 23.28. | B2C2 | 286 |
| 23.29. | Binance | 287 |
| 23.30. | Bitfinex | 289 |
| 23.31. | Bitflyer | 291 |
| 23.32. | BitHumb Pro (Global) | 293 |
| 23.33. | BitMEX | 295 |
| 23.34. | Bitstamp | 297 |
| 23.35. | Bittrex | 299 |
| 23.36. | Blockfills | 301 |
| 23.37. | CoinAPI | 302 |
| 23.38. | Coinbase Pro | 302 |
| 23.39. | CoinMarketCap | 305 |
| 23.40. | CryptoFinance | 305 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.3</td>
<td>Strategy Monitoring</td>
<td>388</td>
</tr>
<tr>
<td>H.4</td>
<td>Installation &amp; Startup</td>
<td>388</td>
</tr>
<tr>
<td>I.</td>
<td>Example Strategy &quot;Short Strangle&quot;</td>
<td>390</td>
</tr>
<tr>
<td>I.1</td>
<td>Trading Idea</td>
<td>390</td>
</tr>
<tr>
<td>I.2</td>
<td>Implementation</td>
<td>391</td>
</tr>
<tr>
<td>I.3</td>
<td>Installation &amp; Startup</td>
<td>392</td>
</tr>
<tr>
<td>J.</td>
<td>Example Strategy &quot;Dividend Capture&quot;</td>
<td>393</td>
</tr>
<tr>
<td>J.1</td>
<td>Trading Idea</td>
<td>393</td>
</tr>
<tr>
<td>J.2</td>
<td>Implementation</td>
<td>393</td>
</tr>
<tr>
<td>J.3</td>
<td>Installation &amp; Startup</td>
<td>393</td>
</tr>
<tr>
<td>K.</td>
<td>Example Strategy &quot;NLP&quot;</td>
<td>395</td>
</tr>
<tr>
<td>K.1</td>
<td>Trading Idea</td>
<td>395</td>
</tr>
<tr>
<td>K.2</td>
<td>Implementation</td>
<td>395</td>
</tr>
<tr>
<td>K.3</td>
<td>Installation &amp; Startup</td>
<td>396</td>
</tr>
<tr>
<td>L.</td>
<td>Example Strategy &quot;Market Maker&quot;</td>
<td>397</td>
</tr>
<tr>
<td>L.1</td>
<td>Trading Idea</td>
<td>397</td>
</tr>
<tr>
<td>L.2</td>
<td>Implementation</td>
<td>397</td>
</tr>
<tr>
<td>L.3</td>
<td>Installation &amp; Startup</td>
<td>398</td>
</tr>
<tr>
<td>M.</td>
<td>Example Strategy &quot;RSI&quot;</td>
<td>399</td>
</tr>
<tr>
<td>M.1</td>
<td>Trading Idea</td>
<td>399</td>
</tr>
<tr>
<td>M.2</td>
<td>Implementation</td>
<td>399</td>
</tr>
<tr>
<td>M.3</td>
<td>Installation &amp; Startup</td>
<td>399</td>
</tr>
<tr>
<td>N.</td>
<td>Example Strategy &quot;EMA&quot; in Python</td>
<td>401</td>
</tr>
<tr>
<td>N.1</td>
<td>Description</td>
<td>401</td>
</tr>
<tr>
<td>N.1.1</td>
<td>Implementation</td>
<td>401</td>
</tr>
<tr>
<td>O.</td>
<td>Example Strategy &quot;BreakOut&quot; in Python</td>
<td>403</td>
</tr>
<tr>
<td>O.1</td>
<td>Description</td>
<td>403</td>
</tr>
<tr>
<td>O.1.1</td>
<td>Implementation</td>
<td>403</td>
</tr>
<tr>
<td>P.</td>
<td>Example Strategy &quot;EMA&quot; in Python via API</td>
<td>405</td>
</tr>
<tr>
<td>P.1</td>
<td>Description</td>
<td>405</td>
</tr>
</tbody>
</table>
List of Figures

3.1. IntelliJ Run Configurations ................................................................. 28
3.2. Run Configurations ............................................................................. 29
4.1. Strategy Development Process ............................................................. 36
5.1. Back Test Report .................................................................................. 75
6.1. Architecture ......................................................................................... 79
7.1. Entities Overview .................................................................................. 80
7.2. Strategy ................................................................................................ 82
7.3. Securities ............................................................................................... 84
7.4. Orders .................................................................................................. 88
7.5. Account ................................................................................................ 89
7.6. Transaction ............................................................................................ 91
7.7. Position ................................................................................................. 92
7.8. Subscription ........................................................................................... 93
7.9. Exchange ............................................................................................... 94
7.10. Order Preference .................................................................................. 95
11.1. AlgoTrader UI Header ........................................................................ 134
11.2. AlgoTrader UI Toolbar ....................................................................... 134
11.3. AlgoTrader UI Settings ....................................................................... 136
11.4. AlgoTrader UI Notification ................................................................. 137
11.5. AlgoTrader UI Alert List ..................................................................... 137
11.6. AlgoTrader UI Order Table ................................................................. 138
11.7. AlgoTrader UI Manual Order Entry .................................................... 138
11.9. AlgoTrader UI Advanced Order Form ................................................. 139
11.10. AlgoTrader UI Advanced Order Form - Routing section, Smart Routing switched off .................................................. 140
11.11. AlgoTrader UI Advanced Order Form - Routing section, Smart Routing switched on ...................................................... 140
11.12. AlgoTrader UI Advanced Order Form - Routing section, Smart Routing disabled ........................................................... 141
11.13. AlgoTrader UI Advanced Order Form - Routing section, Smart Routing equivalent of Simple Order Type .............................................. 141
11.14. Execution Algo details icon visible in action column in the Orders table .............................................................................. 142
11.15. Execution Algo details modal window with a grid listing all children of a given Execution Algo ......................................................... 142
11.16. RFQ Button displayed in the Order section ........................................ 142
11.17. RFQ Entry Form filled with example data ......................................... 143
11.18. RFQ Responses Window ................................................................... 144
11.19. RFQ Retry .......................................................................................... 145
11.20. AlgoTrader UI Transaction Table ...................................................... 145
11.21. AlgoTrader UI Transaction Entry ...................................................... 146
11.22. AlgoTrader UI Transaction Entry and Fees Entry ............................... 146
11.23. AlgoTrader UI Position Table ............................................................ 147
11.24. AlgoTrader UI Market Data Table ..................................................... 147
11.25. AlgoTrader UI Market Data Subscribe .............................................. 148
11.26. AlgoTrader UI Market Data Unsubscribe ......................................... 148
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.27. AlgoTrader UI Transaction Column Selection</td>
<td>149</td>
</tr>
<tr>
<td>11.28. AlgoTrader UI Column Filter</td>
<td>149</td>
</tr>
<tr>
<td>11.29. AlgoTrader UI Chart Widget</td>
<td>150</td>
</tr>
<tr>
<td>11.30. AlgoTrader UI Chart Widget settings</td>
<td>151</td>
</tr>
<tr>
<td>11.31. AlgoTrader UI Chart Widget - selecting security</td>
<td>154</td>
</tr>
<tr>
<td>11.32. AlgoTrader UI Chart Widget - order book chart</td>
<td>155</td>
</tr>
<tr>
<td>11.33. AlgoTrader UI Chart Widget - aggregated order book chart</td>
<td>155</td>
</tr>
<tr>
<td>11.34. HTML5 Custom Widget Example</td>
<td>157</td>
</tr>
<tr>
<td>15.1. Leverage &amp; Exposure</td>
<td>193</td>
</tr>
<tr>
<td>17.1. Order Status Transitions</td>
<td>207</td>
</tr>
<tr>
<td>18.1. Market Data Event Types</td>
<td>214</td>
</tr>
<tr>
<td>23.1. Market Data Subscriptions 1</td>
<td>275</td>
</tr>
<tr>
<td>23.2. Market Data Subscriptions 2</td>
<td>275</td>
</tr>
<tr>
<td>23.3. Market Data Subscriptions 3</td>
<td>276</td>
</tr>
<tr>
<td>23.4. Market Data Subscriptions 4</td>
<td>277</td>
</tr>
<tr>
<td>23.5. Paper Trading Account 1</td>
<td>278</td>
</tr>
<tr>
<td>23.6. Paper Trading Account 2</td>
<td>278</td>
</tr>
<tr>
<td>23.7. Delayed IB Market Data</td>
<td>279</td>
</tr>
<tr>
<td>24.1. Adaptive Send Child Order Retry Policy</td>
<td>332</td>
</tr>
<tr>
<td>24.2. Adaptive Cancel Child Order Retry Policy</td>
<td>333</td>
</tr>
<tr>
<td>25.1. Combinations and Components</td>
<td>334</td>
</tr>
<tr>
<td>25.2. Combination Example</td>
<td>335</td>
</tr>
<tr>
<td>32.1. Health check results</td>
<td>352</td>
</tr>
<tr>
<td>B.1. Box Trading Ranges</td>
<td>359</td>
</tr>
<tr>
<td>B.2. Box Strategy</td>
<td>360</td>
</tr>
<tr>
<td>B.3. Box States</td>
<td>361</td>
</tr>
<tr>
<td>B.4. Box Strategy Performance</td>
<td>361</td>
</tr>
<tr>
<td>B.5. Box HTML5 Custom Widget Example</td>
<td>363</td>
</tr>
<tr>
<td>C.1. Pair Trading Portfolio ID</td>
<td>369</td>
</tr>
<tr>
<td>C.2. Pairs Trading HTML5 Custom Widget Example</td>
<td>371</td>
</tr>
<tr>
<td>D.1. IPO HTML5 Custom Widget Example</td>
<td>374</td>
</tr>
<tr>
<td>E.1. EMA Strategy Example</td>
<td>378</td>
</tr>
<tr>
<td>H.1. Delta Strategy UI</td>
<td>388</td>
</tr>
<tr>
<td>I.1. Short Strangle</td>
<td>391</td>
</tr>
<tr>
<td>N.1. Python Profit Evolution</td>
<td>402</td>
</tr>
</tbody>
</table>
List of Tables

4.1. Strategy life-cycle phase ................................................................. 64
7.1. Entities ......................................................................................... 80
7.2. Strategy Classes ........................................................................ 83
7.3. Portfolio Value Details ................................................................. 83
7.4. Security Types ............................................................................ 84
7.5. Market Data Types ..................................................................... 86
7.6. Order Classes ............................................................................ 88
7.7. Position Valuation Details ............................................................ 93
7.8. Exchange ..................................................................................... 94
7.9. Order Preference ........................................................................ 95
7.10. Quote Request and Quote ............................................................ 96
7.11. Main Services ............................................................................ 97
7.12. Client Services .......................................................................... 98
8.1. AlgoTrader Server modules .......................................................... 118
8.2. Esper tags .................................................................................. 119
9.1. Cache configuration ................................................................... 129
14.1. Position Currency Attribution .................................................... 189
14.2. Transaction Currency Attribution ............................................... 190
15.1. Bar Data Format ...................................................................... 192
19.1. Tick Data Format .................................................................... 230
19.2. Bar Data Format ...................................................................... 231
22.1. Event topics ............................................................................. 243
22.2. Logon ....................................................................................... 252
22.3. Logout ...................................................................................... 252
22.4. Test Request ............................................................................ 252
22.5. Heartbeat ................................................................................ 253
22.6. Resend Request ...................................................................... 253
22.7. New Order Single ................................................................... 253
22.8. Order Cancel Request ............................................................... 254
22.9. Execution Report .................................................................... 255
23.1. Tradable security types ............................................................... 264
23.2. Order type constraints ............................................................... 265
23.3. Account Adapter Functionality Support ........................................ 266
23.4. B2C2 constraints ..................................................................... 287
23.5. Binance constraints .................................................................. 288
23.6. Supported Functionality .............................................................. 289
23.7. Bitfinex constraints .................................................................. 290
23.8. Supported Functionality .............................................................. 290
23.9. BitFlyer constraints .................................................................. 292
23.10. Supported Functionality ............................................................. 293
23.11. Supported Functionality ............................................................. 295
23.12. Supported Instruments .............................................................. 296
23.13. BitMex constraints ........................................................................................................... 296
23.14. Supported Functionality ..................................................................................................... 297
23.15. BitStamp constraints .......................................................................................................... 298
23.16. Supported Functionality ..................................................................................................... 298
23.17. Supported Functionality ..................................................................................................... 300
23.18. Coinbase Pro constraints ...................................................................................................... 304
23.19. Supported Functionality ..................................................................................................... 304
23.20. CryptoFinance constraints .................................................................................................. 306
23.21. Supported Functionality ..................................................................................................... 309
23.22. Enigma constraints ................................................................................................................ 310
23.23. eToroX constraints ............................................................................................................... 311
23.24. Supported Functionality ..................................................................................................... 312
23.25. Supported Functionality ..................................................................................................... 315
23.26. Supported Functionality ..................................................................................................... 317
23.27. Supported Functionality ..................................................................................................... 320
23.28. Supported Functionality ..................................................................................................... 323
24.1. SlicingOrder ............................................................................................................................. 325
24.2. VWAPOrder ............................................................................................................................ 327
24.3. AdaptiveOrder ......................................................................................................................... 329
26.1. Adapter Spring Profiles .......................................................................................................... 339
30.1. Default Log4j Appenders ....................................................................................................... 348
30.2. Production Log4j Appenders ................................................................................................. 348
32.1. Health check results ................................................................................................................. 352
Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*¹ set. The Liberation Fonts set is also used in HTML editions. If not, alternative but equivalent typefaces are displayed.

1.1. Typographic Conventions

The following typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

System input, including shell commands, file names and paths, and key caps and key-combinations are presented as follows.

```
To see the contents of the file my_next_bestselling_novel in the current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press Enter to execute the command.
```

The above includes a file name, a shell command and a key cap, all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the symbol connecting each part of a key-combination. For example:

```
Press Enter to execute the command.

Press Ctrl-Alt-F1 to switch to the first virtual terminal. Press Ctrl-Alt-F7 to return to the X-Windows session.
```

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph are presented as follows.

```
File-related classes include filesystem for file systems, file for files, and dir for directories. Each class has its own associated set of permissions.
```

Words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles are presented as follows.

```
Choose System → Preferences → Mouse from the main menu bar to launch Mouse Preferences. In the Buttons tab, click the Left-handed mouse check box and click Close to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
```

¹ https://pagure.io/liberation-fonts
To insert a special character into a gedit file, choose Applications → Accessories → Character Map from the main menu bar. Next, choose Search → Find from the Character Map menu bar, type the name of the character in the Search field and click Next. The character sought will be highlighted in the Character Table. Double-click this highlighted character to place it in the Text to copy field and then click the Copy button. Now switch back to the document and choose Edit → Paste from the gedit menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all distinguishable by context.

Note the shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow ‘Select Mouse from the Preferences sub-menu in the System menu of the main menu bar’ approach.

Italics denotes text that does not need to be imputed literally or displayed text that changes depending on circumstance. Replaceable or variable text is presented as follows.

To connect to a remote machine using ssh, type ssh username@domain.name at a shell prompt. If the remote machine is example.com and the username on that machine is john, type ssh john@example.com.

The mount -o remount file-system command remounts the named file system. For example, to remount the home file system, the command is mount -o remount /home.

To see the version of a currently installed package, use the rpm -q package command. It will return a result as follows: package-version-release.

Note the words in italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text entered when issuing a command or for text displayed by the system.

1.2. Pull-quote Conventions

Two commonly multi-line data types are set off visually from the surrounding text.

Output sent to a terminal is presented as follows:

<table>
<thead>
<tr>
<th>books</th>
<th>Desktop</th>
<th>documentation</th>
<th>drafts</th>
<th>mss</th>
<th>photos</th>
<th>stuff</th>
<th>git</th>
</tr>
</thead>
<tbody>
<tr>
<td>books_tests</td>
<td>Desktop1</td>
<td>downloads</td>
<td>images</td>
<td>notes</td>
<td>scripts</td>
<td>svgs</td>
<td></td>
</tr>
</tbody>
</table>

Source-code listings are presented and highlighted as follows:

```java
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient {
    public static void main(String args[]) throws Exception {
```
InitialContext iniCtx = new InitialContext();
Object ref = iniCtx.lookup("EchoBean");
EchoHome home = (EchoHome) ref;
Echo echo = home.create();

System.out.println("Created Echo");

System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
}
}

1.3. Notes and Warnings

Finally, three visual styles are used to draw attention to information that might otherwise be overlooked.

**Warning**

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

**Important**

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won’t cause data loss but may cause irritation and frustration.

**Note**

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but might lead to a missed out on a trick that makes life easier.
Introduction

AlgoTrader is a comprehensive algorithmic trading platform that enables both buy side and sell side trading firms to rapidly develop, simulate, backtest and deploy automated quantitative trading strategies on a single platform. Designed by industry experts, it gives users maximum control over their trading experience. Initially designed for global equities, futures, forex and options, AlgoTrader now fully supports automated trading of Cryptocurrencies. AlgoTrader is an extremely reliable and robust system built on a multi-threaded, memory efficient, highly concurrent architecture. It is optimized for high availability and performance to support uninterrupted trading.

The following links provide general information about the system

1 System overview
2 Demo
3 Trial Version
4 Videos
5 Architecture
6 Screenshots
7 Product Features
8 Product Factsheet
9 List of 3rd party libraries

1 https://www.algotrader.com/product/overview/
2 https://www.algotrader.com/product/demo-system/
3 https://www.algotrader.com/product/demo-system/
4 https://www.algotrader.com/product/video/
5 https://www.algotrader.com/product/architecture/
6 https://www.algotrader.com/product/screenshots/
7 https://www.algotrader.com/features/
8 http://doc.algotrader.com/AlgoTraderFactsheet.pdf
9 https://www.algotrader.com/product/3rd-party-libraries/
Chapter 2.

Chapter 2. CONFIDENTIAL

Installation and Deployment

2.1. Windows Installer

We have an installer process for Windows, which sets up an AlgoTrader capable development environment by installing all thirdparty products required to run AlgoTrader and might save you some time vs. the manual setup of each individual component, as described in the next chapter. Please ask your account manager for details if you want to make use of it.

It installs under a directory of your choosing

- An OpenJDK version compatible with AlgoTrader
- The AlgoTrader Server and Example Strategies
- IntelliJ IDEA
- PyCharm CE IDE
- Python 2.7 and 3.7
- The Pip package installer for Python
- MySQL Database
- InfluxDB Database
- dbForge Studio Express
- Interactive Brokers Trader Workstation
- Notepad++

You can opt out of some of these at installation time and the installed software should be sandboxed and will not interfere with other installations of the same software you might have on your machine.

It will also setup and start MySQL and InfluxDB as a service.

In addition you will need a modern browser to display the AlgoTrader UI and if you want to visualise backtest results, also MS Excel.

Note

With the installer, automatic updates are disabled for IntelliJ IDEA, PyCharm CE IDE, dbForge Studio Express and Interactive Brokers Trader Workstation. You should change the update settings on these applications if you intend to use the system for a longer period of time.
Note

The installer does not include Git so the AlgoTrader version is a copy of the current release (bootstrap) and example strategies (examples), the base directory being on your desktop. If you want to update the AlgoTrader version later, you will need to clone the AlgoTrader repository instead.

Note

The installer configures trial user Maven credentials if you do not yet have a Maven configuration on your PC. You should change the credentials to your personal login in the Maven settings file (.m2/settings.xml under your Windows user).

After the installation, start the IntelliJ IDEA using the Idea for Algotrader desktop icon and wait for the examples project to be read.

To be able to run AlgoTrader, you first need to run a Maven install: at the bottom right of the screen, press Enable auto-import, then install the Maven project Bootstrap in the Maven menu on the right, then repeat with the AlgoTrader Example Strategies.
Once that has completed (you should see the log BUILD SUCCESS), you are ready to use AlgoTrader on your machine.

2.2. Development Environment Installation

2.2.1. Prerequisites

Note

It is generally recommended not to use paths with spaces for any of the components used by AlgoTrader (e.g. C:\Program Files).

Java JDK

Install the latest Java JDK 1.8 from Oracle¹ or OpenJDK 1.8 from Red Hat².

Important

AlgoTrader requires Java 1.8.x. Do not use Java 1.9 or greater.

It is necessary to have a Java JDK (Java Development Environment), a Java JRE (Java Runtime Environment) will not be sufficient.

Please set the JAVA_HOME environment variable to point at the directory where the Java JDK is installed. You also need to add JAVA_HOME\bin to your PATH variable. Setup java environment variables.³

---

¹ https://www.oracle.com/java/technologies/javase-jdk8-downloads.html
² https://developers.redhat.com/products/openjdk/download
Maven

AlgoTrader uses Apache Maven for handling of dependencies. The IntelliJ IDEA from the Algotrader installer already has an embedded Maven installation integrated. In case you want to use Maven from the command line, it is necessary to download and install the latest version of Maven and setup it’s environment variables according to the link *Maven setup*.

In particular, please set the **MAVEN_HOME** environment variable to point at the directory where Maven is installed. You also need to add **MAVEN_HOME\bin** to your PATH variable.

All AlgoTrader artifacts are located on our Nexus server which is password protected:

https://repo.algotrader.com/nexus/

It is necessary to create the following file `<user-home>\m2\settings.xml` below content there, to make sure Maven can access our Nexus server. Folder .m2 should have been automatically created while running maven for the first time.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>algotraderrepo</id>
      <username>myusername</username>
      <password>mypassword</password>
    </server>
    <server>
      <id>archetype</id>
      <username>myusername</username>
      <password>mypassword</password>
    </server>
  </servers>
  <profiles>
    <profile>
      <id>algotrader</id>
      <repositories>
        <repository>
          <id>algotraderrepo</id>
          <url>https://repo.algotrader.com/nexus/repository/general/</url>
        </repository>
        <repository>
          <id>archetype</id>
          <url>https://repo.algotrader.com/nexus/repository/general/</url>
        </repository>
      </repositories>
    </profile>
  </profiles>
</settings>
```

* http://maven.apache.org/install.html
<repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>algotrade-repo</id>
      <url>https://repo.algotrader.com/nexus/repository/general/</url>
    </pluginRepository>
  </pluginRepositories>
</repositories>

Note

Please replace myusername and mypassword (both appear twice!) with the username and password provided when licensing AlgoTrader.

Git

AlgoTrader uses Git as its source code management system.

In case one wants to use Git from the command line it is necessary to download and install the latest version of Git.

On Windows use Tortoise Git in combination with Git for Windows.

MySQL

Download and install MySQL Community Server.

Important

The minimum required MySQL version is 5.7.8

Per default the system uses the user name root and password password. To change username and/or password the following properties need to be updated inside conf-core.properties. Alternatively the properties can be changed via Section 2.4, “VM Options”:

# database user name

---

5 https://tortoisegit.org
6 https://gitforwindows.org/
7 https://dev.mysql.com/downloads/
You can create the root user/set the DB password using the following command:

```
mysqladmin -u myUsername password myPassword
```

To work with MySQL it is recommended to install a MySQL client. There are many different MySQL clients available to choose from:

- **devart dbForge Studio Express**\(^8\) (free)
- **MySQL Workbench**\(^9\) (free)
- **SQLyog MySQL**\(^10\) (commercial)

### Note

The Java MySQL JDBC driver sometimes has issues connecting to the MySQL database depending on the MySQL time zone setting. Java Exceptions like the following are an indication for this issue:

```
java.sql.SQLException: The server time zone value 'Coordinated Universal Time' is unrecognized or represents more than one time zone. You must configure either the server or JDBC driver (via the serverTimezone configuration property) to use a more specific time zone value if you want to utilize time zone support.
```

To fix the issue it is recommended to change the MySQL time zone setting by executing the following MySQL statement:

```
SET GLOBAL time_zone = 'UTC' ;
```

In case you get an error like this:

```
mysql> SET GLOBAL time_zone = 'UTC' ;
ERROR 1298 (HY000): Unknown or incorrect time zone: 'UTC'
```

---

\(^8\) [https://www.devart.com/dbforge/mysql/studio/download.html](https://www.devart.com/dbforge/mysql/studio/download.html)

\(^9\) [https://www.mysql.com/products/workbench/](https://www.mysql.com/products/workbench/)

\(^10\) [https://www.webyog.com/](https://www.webyog.com/)
This means that your MySQL database does not know the UTC time zone yet. Follow the instructions under *MySQL Time Zone Issues*\(^\text{11}\) to install them. For Windows this means to download the 5.7 .sql file with the posix time zone definitions and manually loading them into your local DB instance. For Linux this can be easily fixed by running following command:

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

To be sure you do not lose this information, e.g. on reboot, you can set a time zone in the MYSQL my.cnf or my.ini file:

```
[mysqld]
default-time-zone=+00:00
explicit_defaults_for_timestamp=1
```

### InfluxDB

This is an optional component that is used to store historical data. In back testing it is possible to use CSV files to provide historical data as an alternative to using InfluxDB.

**Linux/MacOS:** Download the latest version of *InfluxDB*\(^\text{12}\) and install it according to the *InfluxDB installation instructions*\(^\text{13}\)(Install it in a directory/tree without spaces in the names).

**Windows:** Download version 1.5.2 of *InfluxDB*\(^\text{14}\) and unpack the file in a directory/tree without spaces in the names.

**Important**

On Windows AlgoTrader requires InfluxDB 1.5.2. Do not use InfluxDB 1.6.0 or greater.

To install InfluxDB as a windows service please follow these steps:

- download *nssm*\(^\text{15}\)
- unpack nssm
- update the following sections inside `influxdb.conf` by replacing `<username>` with the username that will run InfluxDB

---

\(^{11}\) [https://dev.mysql.com/downloads/timezones.html](https://dev.mysql.com/downloads/timezones.html)

\(^{12}\) [https://portal.influxdata.com/downloads/](https://portal.influxdata.com/downloads/)

\(^{13}\) [https://docs.influxdata.com/influxdb/v1.5/introduction/installation/](https://docs.influxdata.com/influxdb/v1.5/introduction/installation/)

\(^{14}\) [https://dl.influxdata.com/influxdb/releases/influxdb-1.5.2_windows_amd64.zip](https://dl.influxdata.com/influxdb/releases/influxdb-1.5.2_windows_amd64.zip)

\(^{15}\) [https://nssm.cc/download](https://nssm.cc/download)
Prerequisites

- Go to nssm installed folder, choose win64 or win32 folder, start a command prompt and type:

```bash
.nssm.exe install InfluxDB <full-path-to-influxd.exe> -config <full-path-to-influxdb.conf>
```

(put path in quotation marks if it contains spaces), Example:

```bash
nssm.exe install InfluxDB c:\AlgoTrader\influxdb-1.5.2\influxd.exe -config c:\AlgoTrader\influxdb-1.5.2\influxdb.conf
```

Make sure you run the influxd.exe command, not influx.exe.

- start InfluxDB service via Windows Service Manager

- Add an environment variable named `HOME` pointing to the directory where InfluxDB is installed

All InfluxDB related settings are available within the file `conf-influxdb.properties`

Connection settings

```properties
influxDB.host = localhost
influxDB.port = 8086
influxDB.dbName = algotrader
influxDB.username = 
influxDB.password = 
```
Depending on machine performance and dataset size it may be needed to adjust InfluxDB timeout periods (especially influxDB.readTimeoutMs property) to bigger values.

Per default username/password authentication is disabled. To set username and password based authentication please visit the InfluxDB Authentication and Authorization guide\textsuperscript{16}.

You can manage your InfluxDB data using AlgoTrader’s Historical Data Manager Section 11.3, “Historical Data Manager”

There are several 3rd party client options available to access InfluxDB:

- InfluxDB Chronograf\textsuperscript{17}
- Grafana\textsuperscript{18}

\section*{2.2.2. AlgoTrader Server Code Installation}

The AlgoTrader server code can be installed either via command line or via IDE

\subsection*{2.2.2.1. Command Line}

To install the AlgoTrader server code via command line please perform the following steps.

\subsection*{2.2.2.1.1. Git Checkout}

If one hasn’t installed git, please refer to git installation in chapter Section 2.2.1, “Prerequisites”\textsuperscript{16, 17, 18}

\textsuperscript{16} https://docs.influxdata.com/influxdb/v1.5/query_language/authentication_and_authorization/
\textsuperscript{17} https://docs.influxdata.com/chronograf/v1.5/
\textsuperscript{18} https://grafana.com/docs/features/datasources/influxdb/
Perform a Git clone from the command line:

```
    git clone https://gitlab.algotrader.com/main/Bootstrap.git
```

**Note**

user name and password will be provided when signing up for an AlgoTrader license

**Note**

Please make sure the Maven settings file is updated according to Section 2.2.1, “Prerequisites”.

**2.2.2.1.2. Maven Build**

Execute the following maven command to build all maven projects

```
    mvn clean install
```

**Note**

When running the build process for the first time, this will take a few minutes since all maven dependencies have to be downloaded.

**2.2.2.2. IntelliJ IDEA**

To build AlgoTrader from within IntelliJ IDEA please follow this process.

**2.2.2.2.1. Git Clone**

- On the main menu, select VCS / Git / Clone
- Set the following URL [https://gitlab.algotrader.com/main/Bootstrap.git](https://gitlab.algotrader.com/main/Bootstrap.git), configure the directory where you want the project to be stored and press clone.
- Enter User and Password (provided when licensing AlgoTrader)
- Click OK
- Confirm you want to open the project, e.g. in a new Window.
- Click on Project on the top of the sidebar on the left and you will see the Bootstrap project in the package explorer, with the sub-projects conf and launch.
2.2.2.2. Maven Build

To generate the code click on **Maven** on the top of the sidebar on the right to get the Maven menu, then press the **Execute Maven Goal** button at the top right and select (double-click) **Maven install**. This will download the missing libraries and compile AltoTrader.

![Maven Build Screen](image)

**Run Anything**

```
mvn
```

**Maven Goals**

- mvn clean
- mvn compile
- mvn deploy
- **mvn install**

2.2.3. Python installation

Clients who would like to develop strategies in Python and do not use the AlgoTrader AWS image, will also need to install additional packages.

Python packages can be downloaded as a source distribution via our secured Nexus repository and can be installed via pip:

```
pip3 install algotrader_com --extra-index-url https://repo.algotrader.com/nexus/repository/python/simple
```

for the latest version or if you're looking for a specific version (e.g. 6.0.2)

```
pip3 install algotrader-com==6.0.2.0 --extra-index-url https://repo.algotrader.com/nexus/repository/python/simple
```
Required dependencies are installed automatically. The interface is not distributed via the public PyPI repository. The package functionality has been tested with Python 2.7 and Python 3.7.

The AlgoTrader Python Interface package name is `algotrader_com` to prevent collisions with a non-affiliated library with name `algotrader` that is present on PyPI.

### 2.2.4. Next Steps

After your IDE has been installed please continue with one of the following steps:

- Start AlgoTrader according to *Chapter 3, Starting AlgoTrader*
- Start an example trading strategy according to the Appendix of this document
- Create a trading strategy according to *Chapter 4, Strategy Development*

### 2.3. Server Environment Installation

AlgoTrader uses *Docker*\(^\text{19}\) for server-side installations.

Docker allows packaging of applications with all of its dependencies into a standardized unit for software development.

At the core of the Docker platform is Docker Engine, a lightweight runtime and robust tooling that builds and runs Docker containers.

For an in-depth description of Docker please visit the *What is Docker*\(^\text{20}\) page.

To get started with Docker please visit the *Docker Engine Documentation*\(^\text{21}\) page.

#### 2.3.1. Docker based Installation

Docker is supported on Linux, Windows, OS X as well as different cloud services (e.g. Amazon AWS or Digital Ocean).

Please follow these setups to setup a Docker based AlgoTrader installation:

Install Docker Engine

according to the *Docker Engine installation instructions*\(^\text{22}\)

---

\(^\text{19}\) https://www.docker.com/
\(^\text{20}\) https://www.docker.com/why-docker
\(^\text{21}\) https://docs.docker.com/engine/
\(^\text{22}\) https://docs.docker.com/engine/installation/
Install Docker Compose
   according to the Docker Compose installation instructions\(^{23}\)

\[\text{Note}\]
On Mac and Windows please install the Docker Toolbox that contains Docker Compose

Copy docker compose file
   Copy the following file to the server and make changes as necessary according to Section 2.3.3, “Docker Compose”:

Login to Nexus
   Login to the Docker Repository with the username and password provided when licensing AlgoTrader.
   \[\text{docker login docker.algotrader.com}\]

Run docker compose
   Invoke the following command inside the directory where the docker-compose.yml file is located:
   \[\text{docker-compose up \(-d\)}\]

Open the AlgoTrader UI
   Open the browser and point it to the following URL using the IP retrieved in the previous step.
   \[\text{http://localhost:9090}\]

The above process will setup an AlgoTrader based system made up of the following Docker containers:

- AlgoTrader server
- Interactive Brokers Gateway
- MySQL (with the AlgoTrader MySQL sample data populated)

To startup AlgoTrader with different startup options please visit this chapter on starting AlgoTrader in a Section 3.3, “Server Environment”

To startup one of the AlgoTrader example strategies please visit the appendix of this documentation.

\[\text{2.3.2. Docker Containers}\]

A typical Docker based AlgoTrader installation is made up of the following Docker containers that can be configured via a docker-compose.yml file.

\(^{23}\) https://docs.docker.com/compose/install/
\(^{24}\) https://gitlab.algotrader.com/main/Bootstrap/tree/master/launch/docker-compose.yml
2.3.2.1. AlgoTrader Container

The AlgoTrader code is located in the directory `/usr/local/algotrader` inside the Docker container.

The following Docker environment variables are relevant for the AlgoTrader container:

- **ALGOTRADER_HOST** (default: `algotrader`). This variable is used by strategies to reference the AlgoTrader Docker container.
- **DATABASE_HOST** (default: `mysql`)
- **DATABASE_PORT** (default: `3306`)
- **DATABASE_NAME** (default: `algotrader`)
- **DATABASE_USER** (default: `root`)
- **DATABASE_PASSWORD** (default: `password`)
- **IB_GATEWAY_HOST** (default: `ibgateway`)

**IB_GATEWAY_ACCOUNT** optional, only needs to be specified if the IB login has multiple accounts associated
- **INFLUXDB_HOST** (default: `influxdb`)
- **VM_ARGUMENTS** Additional VM Options to be added to the java process (e.g. `-DlogLevel=INFO`)
- **SPRING_PROFILES** (default: `live,pooledDataSource,iBMarketData,iBNative,iBHistoricalData,embeddedBroker,html5`) Spring profiles to be used (comma separated list)
- **STARTER_CLASS** (default: `ch.algotrader.starter.ServerStarter`)
- **SECRET_FILE** Secret file name (e.g. `/run/secrets/api_keys`)

In addition the command line switch `-i` can be used to load the MySQL sample data file (`samples/db/mysql/mysql-data.sql`) on first start up. The sample data will only be loaded if the security table contains no data.

To use the `-i` switch please use the following directive inside `docker-compose.yml`:

```yaml
command: -i
```

The AlgoTrader Docker container will run through the following process on startup:

1. Wait for MySQL to be available. When starting up MySQL and AlgoTrader at the same time (using docker compose) it will take the database a few seconds to become available.
2. Run all Flyway migrate scripts (see Section 10.2, “Flyway”).
3. Load MySQL sample data if the `-i` command line switch is used (see above).
4. Start the AlgoTrader server

2.3.2.2. AlgoTrader Strategy Containers

AlgoTrader based trading strategies run in separate Docker containers when running in distributed mode. When running a single strategy in embedded-mode the strategy will run inside the same Docker container as the AlgoTrader server.

The strategy code is located in the directory `/usr/local/strategy` inside the Docker container.

All strategy Docker containers are based on the AlgoTrader Docker container, so environment variables from the AlgoTrader docker container can be reused inside strategy containers.

To build a strategy Docker container the following `Dockerfile` has to be added to the root of the project:

```
FROM docker.algotrader.com/algotrader/algotrader:latest

ENV STRATEGY_NAME=XYZ

WORKDIR /usr/local/strategy
ADD target/*.jar lib/

ENTRYPOINT ["/usr/local/algotrader/bin/docker-strategy-run.sh"]
CMD ["-e"]
```

Please replace `XYZ` with the name of the strategy.

Strategy Docker containers use the `/usr/local/algotrader/bin/docker-strategy-run.sh` startup script that is provided by the AlgoTrader Docker container.

The startup script supports both embedded and distributed mode, see: Section 3.2, “Live Trading Mode”

To start the strategy Docker container in embedded mode please use the `-e` command line switch inside the `docker-compose.yml` file of your strategy:

```
command: -e
```

This will cause the system to run through the following process:

1. Wait for MySQL to be available. When starting up MySQL and AlgoTrader at the same time (using docker compose) it will take the database a few seconds to become available.

2. Run all Flyway migrate scripts (see Section 10.2, “Flyway”).

3. Load MySQL data from `db/mysql/mysql-data.sql`. MySQL is only loaded if the entry in the strategy table for `STRATEGY_NAME` is missing.

4. Start the strategy in embedded mode
To start the strategy Docker container in distributed mode please use the `-d` command line switch inside the `docker-compose.yml` file of your strategy:

```
command: -d
```

This will cause the system to run through the following process:

1. Wait for the AlgoTrader RMI port (1199) to be available
2. Wait for the ActiveMQ JMS port (61616) to be available
3. Start the strategy in distributed mode

When running the system in distributed mode the AlgoTrader server needs to be run in a separate Docker container. Since trading strategies do not have access to the database directly MySQL data needs to be loaded manually by connecting to the database with a MySQL client. It is therefore suggested to follow this process when starting up the system in distributed mode:

1. Startup MySQL, IB Gateway and AlgoTrader

```
docker-compose up -d mysql ibgateway algotrader
```

Please see the following chapter about changing IB API settings (Read-Only API)

2. Load MySQL data by connecting a MySQL client to port 3306 of the Docker Engine

3. Start strategies

```
docker-compose up -d XYZ
```

Please replace `XYZ` with the name of the strategy.

### 2.3.2.3. Interactive Brokers Gateway

This container is made up of the following two components:

- Interactive Brokers **IB Gateway**
- **IB Controller** which allows running IB Gateway in an automated fashion

26 https://github.com/ib-controller/ib-controller
The following environment variables are relevant for the IB Controller container:

- **TWS_USERNAME** *(default: pmdemo)*
- **TWS_PASSWORD** *(default: demouser)*
- **TRADING_MODE** *(default: paper)*

To run IB Gateway on a headless server (i.e. the Docker container) an `xvfb` virtual frame buffer is used.

Unfortunately only few settings of the IB Gateway can be managed via the IB Controller. All other settings have to be managed via the IB Gateway UI itself which is not visible on the Docker container.

This is especially cumbersome for the Read-Only API trading mode that the is set by default. If this mode is active, placement of orders is not allowed.

To change any of the IB Gateway settings (e.g. Read-Only API trading mode) please execute the following steps:

1. The IB Gateway container stores IB settings inside a *Docker Volume*. This volume can be mapped to a local directory as follows.

   **On Linux and Mac**

   ```
   volumes:
   - /var/lib/tws:/var/lib/tws
   ```

   This will make IB Gateway settings available in the local directory `/var/lib/tws`

   **On Windows**

   ```
   volumes:
   - c:/Users/Administrator/Documents/tws:/var/lib/tws
   ```

   This will make IB Gateway settings available in the local directory `c:\Users\Administrator\Documents\tws`

2. Install and start IB Gateway on a regular workstation (Windows, Mac or Linux)

3. Go to Configure / Settings / API / Settings

4. Make necessary changes (e.g. deselect the Read-Only API check box) and click OK

---

28 [https://docs.docker.com/storage/volumes/](https://docs.docker.com/storage/volumes/)
5. Close the IB Gateway

6. Inside the IB Gateway installation folder there will be one or multiple sub-directories starting which have a name made up of 8-9 characters starting with a the letter d. Please select the directory with the latest timestamp and makes sure it contains a file named ibg.xml

7. Copy this directory (e.g. darykwqzr) into the IB Gateway settings directory linked above:

8. Copy the jts.ini file into the IB Gateway settings directory linked above:

9. Start the IB Gateway Docker Container:

```
docker-compose create start ibgateway
```

**Note**
The above steps will not work for the public pmdemo account which gets reset upon each startup.

### 2.3.2.4. MySQL

MySQL provides a fully configured Docker container. For further details please visit [MySQL on Docker Hub](https://hub.docker.com/_/mysql/).

The following environment variables are relevant for the MySQL container:

- `MYSQL_ROOT_PASSWORD` *(default: password)*
- `MYSQL_DATABASE` *(default: algotrader)*

MySQL data is stored inside a [Docker Volume](https://docs.docker.com/storage/volumes/). This volume can be mapped to a local directory as follows.

**On Linux and Mac**

```
volumes:
  - /var/lib/mysql:/var/lib/mysql
```

This will make MySQL data available in the local directory `/var/lib/mysql`

**On Windows**

```
volumes:
  - c:/Users/Administrator/Documents/mysql:/var/lib/mysql
```

---

29 https://hub.docker.com/_/mysql/
30 https://docs.docker.com/storage/volumes/
This will make MySQL data available in the local directory `c:\Users\Administrator\Documents\mysql`.

### 2.3.2.5. InfluxDB

InfluxDB provides a fully configured Docker container. For further details please visit the [InfluxDB on Docker Hub](https://hub.docker.com/_/influxdb/)

InfluxDB data is stored inside a **Docker Volume**. This volume can be mapped to a local directory as follows.

#### On Linux and Mac

```yaml
volumes:
  - /var/lib/influxdb:/var/lib/influxdb
```

This will make MySQL data available in the local directory `/var/lib/mysql`.

#### On Windows

```yaml
volumes:
  - c:/Users/Administrator/Documents/influxdb:/var/lib/influxdb
```

This will make MySQL data available in the local directory `c:\Users\Administrator\Documents\influxdb`.

### 2.3.3. Docker Compose

Docker based applications typically consist of many small applications that work together. Docker transforms these applications into individual containers that are linked together. Instead of having to build, run and manage each individual container, Docker Compose allows definition of multi-container application with all of its dependencies in a single file, then startup the application in a single command. The application's structure and configuration are held in a single place, which makes starting up applications simple and repeatable everywhere.

For further details regarding Docker Compose please visit the [Docker Compose documentation](https://docs.docker.com/compose/).

Docker Compose uses `docker-compose.yml` files to configure multi-container applications. AlgoTrader ships with a default `docker-compose.yml` file located inside the top-level AlgoTrader project directory:

```yaml
algotrader:
  image: docker.algotrader.com/algotrader/algotrader
  command: -i
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...
  links:
```

---

31 [https://hub.docker.com/_/influxdb/](https://hub.docker.com/_/influxdb/)
32 [https://docs.docker.com/storage/volumes/](https://docs.docker.com/storage/volumes/)
33 [https://docs.docker.com/compose/](https://docs.docker.com/compose/)
Docker Compose

- mysql
- ibgateway
- influxdb

ports:
  - 9090:9090
  - 61614:61614

ibgateway:
  image: docker.algotrader.com/interactivebrokers/ibgateway
  environment:
    TWS_USERNAME: pmdemo
    TWS_PASSWORD: demouser
  volumes:
    - /var/lib/tws

mysql:
  image: mysql:5.7.22
  environment:
    MYSQL_ROOT_PASSWORD: password
    MYSQL_DATABASE: algotrader
  ports:
    - 3306:3306
  volumes:
    - /var/lib/mysql

influxdb:
  image: influxdb:1.5.2
  ports:
    - 8086:8086
  volumes:
    - /var/lib/influxdb

Using this docker-compose.yml file will create Docker containers for AlgoTrader, IB Gateway, MySQL and InfluxDB. The following items are present in the file:

- service name: e.g. algotrader, ibgateway
- image: the name of the image to use in the format namespace/repository:version (e.g. docker.algotrader.com/algotrader/algotrader:latest)
- command: command line argument to pass to the Docker container (e.g. -i). See Section 2.3.2, “Docker Containers” for supported command line arguments
- links: services to link to this container (e.g. MySQL & ibgateway). Adding a link will make the target container accessible with the correct IP by its link name
- ports: ports to map on the host machine. E.g. 3306:3306 will map port 3306 of the MySQL container to 3306 of the host machine
• environment: environment variables to use. See Section 2.3.2, “Docker Containers” for supported environment variables

• volumes: Docker Data volumes to create. E.g. c:/Users/mysql:/var/lib/mysql will map the directory /var/lib/mysql inside the container to c:\mysql on the host machine

2.3.3.1. Passwords

The example docker-compose.yml file specifies multiple passwords directly. For security purposes this often not advisable. As an alternative passwords can be stored using Docker secrets

2.3.3.2. Config Files

All properties within the AlgoTrader *.properties files can be overwritten using the docker environment variable VM_ARGUMENTS as mentioned in Section 2.3.2.1, “AlgoTrader Container”.

If a large number of properties need to be changed or if other config files need to be changed (e.g. fix.cfg) it is recommended to follow this process:

1. start a local instance of the AlgoTrader container using the command

   ```bash
docker run -i -t --entrypoint /bin/bash docker.algotrader.com/algotrader/algotrader
root@9f9adac97f3c:/usr/local/algotrader#
```

   Take note of the container ID that has been created, 9f9adac97f3c

2. make changes to any of the config files inside /usr/local/algotrader/bootstrap/conf

3. Exit the modified AlgoTrader container using exit

4. Commit the change to the modified AlgoTrader container using the following command

   ```bash
docker commit CONTAINER_ID docker.algotrader.com/algotrader/algotrader:VERSION
```

   using the CONTAINER_ID retrieved above and setting a new VERSION

It is also possible to override the entire /usr/local/algotrader/bootstrap/conf file using Docker volumes.

It can be done by putting additional volume to algotrader container in docker-compose.yml file:

```yaml
volumes:
  - ./conf:/usr/local/algotrader/bootstrap/conf/
```

And copying all necessary config files (fix.cfg included) to the created ./conf folder on local machine.

---

34 https://docs.docker.com/engine/swarm/secrets/
35 https://docs.docker.com/storage/volumes/
2.3.3.3. Encrypting sensitive configuration with Docker Secrets

For security reasons, it is recommended to store sensitive configuration (e.g. adapter API key and API secret) in encrypted form.

Docker provides a solution to so called docker secrets. [Docker Secrets documentation](https://docs.docker.com/engine/swarm/secrets/)

To store config values as docker secrets one needs to follow these steps:

1. prepare the temporary file `myKeys.txt` with the sensitive data

   For example by copying the following properties from `conf-bmx.properties`:

   ```properties
   bmx.apiKey=revdPMrxxxxxxxxxxfdsPo
   bmx.apiSecret=fFzYObWcVlyyyyyyyyyFmgXk
   ```

2. create a new secret `api_keys` using docker secret command

   ```bash
   docker secret create api_keys myKeys.txt
   ```

3. remove the temporary file `myKeys.txt`

4. use the following `docker-compose.yml` with references to the `api_keys` secret

   Note the additional section `secrets:` at the bottom of the file:

   ```yaml
   version: '3.1'

   services:

   algotrader:
     image: docker.algotrader.com/algotrader/algotrader
     command: -i
     environment:
       SPRING_PROFILES:
         live,pooledDataSource,bMX,bMXMarketData,bMXAccount,embeddedBroker,html5
     SECRET_FILE: /run/secrets/api_keys
     secrets:
       api_keys
     deploy:
       restart_policy:
         condition: none
     ports:
       - 9090:9090
   ```

---

36 [docs.docker.com/engine/swarm/secrets/](https://docs.docker.com/engine/swarm/secrets/)
lbateway:
  image: docker.algotrader.com/interactivebrokers/lbateway
  environment:
    TWS_USERNAME: pmdemo
    TWS_PASSWORD: demouser
  volumes:
    - /var/lib/tws
  depends_on:
    - algotrader

mysql:
  image: mysql:5.7.22
  environment:
    MYSQL_ROOT_PASSWORD: password
    MYSQL_DATABASE: algotrader
  ports:
    - 3306:3306
  volumes:
    - /var/lib/mysql
  depends_on:
    - algotrader

influxdb:
  image: influxdb:1.5.2
  ports:
    - 8086:8086
  volumes:
    - /var/lib/influxdb
  depends_on:
    - algotrader

secrets:
  api_keys:
    external: true

---

Note

Docker secrets as shown above are only available to swarm services, not to standalone containers. To use this feature, algotrader must run as a service with a scale of 1, on a single node cluster.

Prior to creating and using secrets initialize machine as swarm manager using following command:

docker swarm init
Operating in Swarm mode means using different docker commands, see examples below.

To deploy and run a services use `docker stack` instead of `docker-compose`:
```
docker stack deploy --compose-file=docker-compose.yml mystack
```

To look at logs:
```
docker service logs -f mystack_algotrader
```

Also, the algotrader network has to be created with swarm scope:
```
docker network create algotrader --scope swarm --driver overlay
```

Otherwise the docker swarm will not be able to use it and the algotrader image will not start. To create Docker secret from e.g. `myKeys.txt` file use:
```
docker secret create api_keys myKeys.txt
```

Please also refer to Section 27.1.1, “Encrypting sensitive configuration values”

### 2.3.3.4. Log Files

It is a Docker best-practice to have only one running process per Docker container. Typically application output is logged directly to the console where it can be viewed using the command `docker logs` according to the [Docker documentation](https://docs.docker.com/engine/reference/commandline/logs/).

Sometimes this is not enough and one wishes to write additional information (e.g. fix messages) to a separate log file. To get access to this log file from outside the container it is advised to create an additional volume:
```
volumes:
  - ~/fix.log:/usr/local/algotrader/logs/fix.log
```

### 2.3.4. Docker Management

In addition to using the Docker command line, several options exist for management of docker based installations

#### 2.3.4.1. Portainer

Portainer is another alternative Docker web interface.

---

37 [https://docs.docker.com/engine/reference/commandline/logs/](https://docs.docker.com/engine/reference/commandline/logs/)
To use Docker UI please add the following to the `docker-compose.yml` file:

```yaml
docker-ui:
  image: portainer/portainer
  command: -H unix:///var/run/docker.sock
  ports:
    - 9000:9000
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
```

For further details please visit [Docker UI on Docker Hub](https://hub.docker.com/r/dockerui/dockerui/) and [Docker UI on GitHub](https://github.com/kevana/ui-for-docker).

### 2.3.4.2. Kitematic

When running Docker on Windows or Mac [Docker Kitematic](https://kitematic.com/) provides a UI for management of the Docker engine.
2.4. VM Options

Many characteristics of the system can be customized with VM Options, the following list provides an overview of commonly used VM Options.

- **-DlogLevel**
  
  log4j log level ({ERROR, WARN, INFO or DEBUG})

- **-Dspring.profiles.active**

  list of Spring Profiles to activate (see Section 26.2, "Spring Profiles")

- **-Xmx**

  increase the Java Heap Size to specified amount (e.g. 2048M)

AlgoTrader specific configuration parameters can be changed inside the .properties files. As an alternative configuration parameters can also be provided as VM Options in which case they will overwrite existing parameters inside *.properties files.

- **-Dstatement.closePosition=false**
Starting AlgoTrader

As a first step you make sure that your AlgoTrader license key is properly configured. The license key was provided in the Email after signing up for the AlgoTrader free 30-day trial or when purchasing an AlgoTrader license. The license key needs to be configured inside the file /algotrader-conf/src/main/resources/conf.properties as follows.

The screenshot below shows where to find this file in the AlgoTrader IntelliJ IDE, for which you have a desktop shortcut if you used the AlgoTrader installer process:

Now AlgoTrader can be started through various Run Configurations which are available to launch the various operation modes of AlgoTrader. To access the available Run Configurations in the AlgoTrader IntelliJ IDEA, follow screenshots below.

Open Run Configurations
Select the downward facing black arrow next to the green run symbol

Figure 3.1. IntelliJ Run Configurations
Select Run Configuration

Select the Run Configuration in the list on the left. The right hand side will show the project the Run Configuration will start in as well as the Main Java class.

Update program and VM options if necessary. You can expand them by pressing the 2 facing arrows on the right.

The run configurations of the AlgoTrader installer have been configured to run with the TWS demo account and have IB delayed market data enabled. If you've got a proper IB account, simply remove the setting that enables delayed market data `-Dib.pricefeed.allowDelayedMarketData=true` from the argument list of the starter by editing the relevant run configuration, expanding the VM options and removing that entry:

```plaintext
-Dib.pricefeed.allowDelayedMarketData=true
```

![Run/Debug Configurations](image)

**Figure 3.2. Run Configurations**

### 3.1. Simulation Mode

To run AlgoTrader in Simulation Mode and perform a back test of a strategy the class `ch.algotrader.starter.SimulationStarter` has to be invoked.

In IntelliJ Run Configurations named `SimulationStarter-simulate-xxx` are provided which contain the following items:

**Main-Tab**

- **Project:** strategy project
- **Main Class:** `ch.algotrader.starter.SimulationStarter`

**Configuration-Tab / Program Arguments**

```
simulateWithCurrentParams
```


Arguments-Tab / VM Options

-Dsimulation=true
-DstrategyName=XXX
-DdataSource.0.dataSetType=TICK
-DdataSource.dataSetName=yyy
-Dspring.profiles.active=simulation,pooledDataSource

Note
To run the strategy in simulation mode a dependency to algotrader-core has to be added to the maven dependencies of the trading strategy. Alternatively one can create a separate maven module containing dependencies to the strategy project as well as algotrader-core.

3.2. Live Trading Mode

To run AlgoTrader in Live Trading Mode the corresponding Adapter (e.g. local broker client, Fix session, VPN Connection, etc.) needs to be up and running.

If using InteractiveBrokers the Trader Workstation or the IB Gateway have to be running with the following configurations under API/Settings:

• Enable ActiveX and Socket Clients
• Read-Only Mode: false
• Socket Port: 4001
• Trusted IP Addresses: 127.0.0.1

When running AlgoTrader in Live Trading Mode the AlgoTrader Server and the Strategies can either be run in separate JVMs (distributed mode) or the entire system can be run within one single JVM (embedded mode).

Before starting AlgoTrader check the database table strategy. The column AUTO_ACTIVATE should be set to true for records corresponding to the AlgoTrader Server and the trading strategy one wants to trade.

3.2.1. Embedded Mode

In Embedded Mode both the AlgoTrader Server and the Strategy run within the same JVM.

Note
Only one Strategy can be run at once in Embedded Mode

In IntelliJ Run Configurations named EmbeddedStrategyStarter-xxx are provided which contain the following items:
3.2.2. Distributed Mode

In Distributed Mode the AlgoTrader Server and the Strategy / Strategies run in separate JVMs and have to be started separately.

To Start the AlgoTrader Server in distributed mode the IntelliJ run configurations ServerStarterXX is provided which contain the following items:

Main-Tab

- Project: algotrader-core
- Main Class: ch.algotrader.starter.ServerStarter

Configuration-Tab / VM Options

- 

 dataSource: the hibernate datasource to use, either: pooledDataSource, singleDataSource

 marketDataProfile: the SpringProfile corresponding to the market data interface in use (e.g. iBMarketData or bBMarketData)

 tradingProfile: the SpringProfile corresponding to the adapter in use (e.g. iBNative or iBFix)

Note

To run the strategy in embedded mode a dependency to algotrader-core has to be added to the maven dependencies of the trading strategy. Alternatively one can create a separate maven module containing dependencies to the strategy project as well as algotrader-core.
• **tradingProfile**: the SpringProfile corresponding to the adapter in use (e.g. `iBNative` or `iBFix`)

To start a Strategy IntelliJ Run Configurations `StrategyStarter-xxx` are provided which contain the following items:

**Main-Tab**

• Project: strategy project

• **Main Class**: `ch.algotrader.starter.StrategyStarter`

**Configuration-Tab / VM Options**

```
-DstrategyName=TEST
-Dspring.profiles.active=live
```

### 3.3. Server Environment

AlgoTrader uses Docker for server environment installations. When using Docker, various components of the system as well as their configurations are managed through `docker-compose.yml` files. As a first step when using The AlgoTrader Docker server environment is to configure the AlgoTrader license key within the `docker-compose.yml` file by replacing the `...` with the license key that was provided in the Email after signing up for the AlgoTrader free 30-day trial or when purchasing an AlgoTrader license.

```
xyz:
  image: xyz
  command: -d
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...

algotrader:
  image: docker.algotrader.com/algotrader/algotrader
  container_name: algotrader
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...
```

### 3.3.1. Embedded Mode

In Embedded Mode both the AlgoTrader Server and the Strategy run within the same JVM.

**Note**

Only one Strategy can be run at once in Embedded Mode

To run the system in embedded mode create a `docker-compose.yml` file similar to the following:
xyz:
    image: xyz
    command: -e
    environment:
        - VM_ARGUMENTS=-Dkeygen.id=...
    links:
        - mysql
        - ibgateway
        - influxdb
    ports:
        - 9090:9090
        - 61614:61614
    environment:
        STRATEGY_NAME: XYZ

ibgateway:
    image: docker.algotrader.com/interactivebrokers/ibgateway
    environment:
        TWS_USERNAME: pmdemo
        TWS_PASSWORD: demouser
    volumes:
        - /var/lib/tws

mysql:
    image: mysql:5.7.22
    environment:
        MYSQL_ROOT_PASSWORD: password
        MYSQL_DATABASE: algotrader
    ports:
        - 3306:3306
    volumes:
        - /var/lib/mysql

influxdb:
    image: influxdb:1.5.2
    ports:
        - 8086:8086
    volumes:
        - /var/lib/influxdb

Please replace xyz/XYZ with the name of the trading strategy. Please refer to Chapter 4, Strategy Development on how to create a new trading strategy.

To start the system in embedded mode please run the following command from the directory where the docker-compose.yml file is located:

docker-compose up -d
This will create the following docker containers: strategy (xyz), ibgateway & mysql

For further details please see Section 2.3.3, “Docker Compose”

3.3.2. Distributed Mode

In Distributed Mode the AlgoTrader Server and the Strategy / Strategies run in separate JVMs and have to be started separately.

To run the system in distributed mode create a docker-compose.yml file similar to the following:

```yaml
xyz:
  image: xyz
  command: -d
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...
  links:
    - algotrader
  environment:
    STRATEGY_NAME: XYZ

algotrader:
  image: docker.algotrader.com/algotrader/algotrader
  container_name: algotrader
  environment:
    - VM_ARGUMENTS=-Dkeygen.id=...
  links:
    - mysql
    - ibgateway
    - influxdb
  ports:
    - 9090:9090
    - 61614:61614

ibgateway:
  image: docker.algotrader.com/interactivebrokers/ibgateway
  container_name: ibgateway
  environment:
    TWS_USERNAME: pmdemo
    TWS_PASSWORD: demouser
  volumes:
    - /var/lib/tws

mysql:
  image: mysql:5.7.22
  container_name: mysql
  environment:
    MYSQL_ROOT_PASSWORD: password
    MYSQL_DATABASE: algotrader
```
Please replace *xyz / XYZ* with the name of the trading strategy.

To start the system in distributed mode please run the following command from the directory where the `docker-compose.yml` file is located:

```
docker-compose up -d
```

This will create the following docker containers: strategy (xyz), algotrader, ibgateway & mysql

For further details please see *Section 2.3.3, “Docker Compose”*

---

**Important**

In the Distributed Mode the strategies communicate with the server via messaging provided by ActiveMQ. Every message has a default TTL (time-to-live) set to 60s configurable via the configuration property `activeMQ.message.ttl` (in ms). This is to prevent the messages from piling up and consuming a lot of memory in case the strategy isn't able to process them in time.
Strategy Development

Warning
It is recommended to perform thorough Simulation / Back Testing of newly developed strategies. After that the strategy should be tested with a Paper Trading Account. At the end of a thorough test procedure, the new strategy can be put into production. At the beginning of live trading it is recommended to use a small trading amount only.

The following diagram shows the general procedure for developing new strategies:

Figure 4.1. Strategy Development Process

4.1. Creating a Trading Strategy

The following paragraph will give a short example based on a simple moving average strategy (with the Short Name EMA).

The following 2 options can assist in creating a new trading strategy, the AlgoTrader Strategy Wizard and the AlgoTrader Maven Archetype.

In addition to this setup, you will need to create a database entry for your strategy. Please refer to the strategy table definition.

4.1.1. AlgoTrader Strategy Wizard

The Strategy Wizard provides an easy way to automatically create all artifacts necessary for an AlgoTrader based trading strategy. Internally the Strategy Wizards makes use of the AlgoTrader Archetype, see
Section 4.1, “Creating a Trading Strategy”. The Strategy Wizard provides options for three different types of Trading Strategies:

- Esper based Strategies
- Java based Strategies (without Esper)
- Simple Strategies, consisting of only one file (without Esper)

The AlgoTrader IntelliJ IDEA from the Section 2.1, “Windows Installer” already has these archetypes configured. To add them to your own IDEA installation, simply create new archetypes with group id `algotrader`, artifact id `algotrader-archetype-esper`, `algotrader-archetype-java` or `algotrader-archetype-simple` and set the version to your AlgoTrader version.

Inside IntelliJ IDEA, the Strategy Wizard can be started via File / New / Module which will bring up the following screen where you should select Maven on the left, check the Create from archetype box and select your desired archetype, e.g. `algotrader-archetype-simple`, then press Next.

Configure the next screen by setting the parent project (e.g. AlgoTrader Example Strategies), the strategy name (e.g. ema), set the group id to `ch.algotrader.strategy` and press Next.
Important

For Spring Auto-Wiring to work the package name needs to be `ch.algotrader.strategy`. If a different package is assigned services (e.g. `OrderService` and `LookupService`) will not be available.

On the following page, add 2 additional name/value pairs using the + button:

- name: your strategy name (e.g. `ema`), all lower-case, no periods, no dashes
- serviceNamne: the name of the strategy service (e.g. `EMA`), first letter upper-case or all upper-case, do not include Service at the end (e.g. do not specify `EMAService`)

When clicking Finish the Strategy Wizard will create a new project named like your strategy.
4.1.2. AlgoTrader Maven Archetype

The AlgoTrader Maven Archetype is a project template that can be used to create a new AlgoTrader trading Strategy. To use the Maven Archetype execute the following command from the command line in a new empty directory.

To create Esper bases strategies execute (replace <version> with the corresponding AlgoTrader version):

```
mvn archetype:generate -DarchetypeGroupId=algotrader -DarchetypeArtifactId=algotrader-archetype-esper -DarchetypeVersion=<version>
```

To create Java strategies execute (replace <version> with the corresponding AlgoTrader version):

```
mvn archetype:generate -DarchetypeGroupId=algotrader -DarchetypeArtifactId=algotrader-archetype-java -DarchetypeVersion=<version>
```

To create simple strategies consisting of only one single java file execute (replace <version> with the corresponding AlgoTrader version):

```
mvn archetype:generate -DarchetypeGroupId=algotrader -DarchetypeArtifactId=algotrader-archetype-simple -DarchetypeVersion=<version>
```

The Maven Archetype will ask for the following input parameters:

- **groupId**: The maven group id (e.g. algotrader), all lower-case, can contain periods
- **artifactId**: The maven artifact id (e.g. algotrader-ema), all lower-case, can contain dashes
- **version**: The maven version (e.g. 1.0.0-SNAPSHOT), x.y.z, plus optionally -SNAPSHOT
- **packageName**: The java package name (ch.algotrader.strategy), all lower-case, can contain periods.
- **name**: The name of the strategy (e.g. ema), all lower-case, no periods, no dashes
- **serviceName**: The name of the strategy service (e.g. EMA), first letter upper-case or all upper-case, do not include Service at the end (e.g. do not specify EMAService)
Important

For Spring Auto-Wiring to work the package name needs to be ch.algotrader.strategy. If a different package is assigned services (e.g. OrderService and LookupService) will not be available.

4.1.3. Generated Artifacts Java Archetype

The Java Archetype will generate the following artifacts:

/src/main/java/ch/algotrader/strategy/EMAService.java
   The strategy service class

/src/main/java/ch/algotrader/strategy/EMAConfig.java
   The strategy configuration class

/src/main/java/ch/algotrader/strategy/Metrics.java
   A sample class that can be used within the strategy

/src/main/java/ch/algotrader/strategy/State.java
   A sample enum that can be used within the strategy

/runConfigurations/*.xml
   IntelliJ IDEA (v.2020.1 or greater) Run Configurations to start the Strategy in embedded mode and simulation mode.

/pom.xml
   The Maven project object model file containing general information about the Trading Strategy

/Dockerfile
   The Docker file

4.1.3.1. EMAService.java

This is the main Java-class containing the Business Logic.

The references to the Services provided by the AlgoTrader Server (e.g. OrderService, PositionService, etc.) will be injected on startup by the Spring Framework

```java
public class EMAService extends ConfigAwareStrategyService<EMAConfig> {

    @Override
    public void onStart(final LifecycleEventVO event) {
        getSubscriptionService().subscribeMarketDataEvent(getStrategyName(), getConfig().getSecurityId());
    }
}
```
The class **EMAService** method contains the following items:

1. Once the strategy has reached the **START** live cycle phase subscribe to the security needed for this strategy.
2. Construct an Order Value Object using the `MarketOrderVOBuilder`. The `OrderVO` contains a reference to the strategy, the security, the account as well as the quantity and the order side.
3. Send the Order to the market via the `OrderService`.

### 4.1.3.2. pom.xml

The Maven `pom.xml` file contains the Maven project definition as well as Maven dependencies:

```xml
  <modelVersion>4.0.0</modelVersion>
  <groupId>algo-trader</groupId>
  <artifactId>algo-trader-ema</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ema strategy</name>

  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```
4.1.3.3. Dockerfile

The Dockerfile contains all relevant information to build a Docker container:

FROM docker.algotrader.com/algotrader/algotrader:latest
ENV STRATEGY_NAME=EMA
WORKDIR /usr/local/strategy
ADD target/*.jar lib/
ENTRYPOINT ["/usr/local/algotrader/bin/docker-strategy-run.sh"]
CMD ["-e"]

4.1.4. Generated Artifacts Simple Java Archetype

The Simple Java Archetype will generate the following artifacts:

/src/main/java/ch/algotrader/strategy/EMAService.java
  The strategy service class

/runConfigurations/*.xml
  IntelliJ IDEA (v.2020.1 or greater) Run Configurations to start the Strategy in embedded mode and simulation mode.

/pom.xml
  The Maven project object model file containing general information about the Trading Strategy
/Dockerfile
The Docker file

4.1.4.1. EMAService.java

This is the main Java-class containing the Business Logic.

The references to the Services provided by the AlgoTrader Server (e.g. OrderService, PositionService, etc.) will be injected on startup by the Spring Framework

```java
public class EMAService extends ConfigAwareStrategyService<EMAConfig> {

    @Override
    public void onStart(final LifecycleEventVO event) {
        getSubscriptionService().subscribeMarketDataEvent(getStrategyName(), getConfig().getSecurityId(), AdapterType.IB.name());
    }

    @Override
    public void onBar(BarVO bar) {

        MarketOrderVO order = MarketOrderVOBuilder.create()
            .setStrategyId(getStrategy().getId())
            .setAccountId(getConfig().getAccountId())
            .setSecurityId(getConfig().getSecurityId())
            .setQuantity(new BigDecimal(getConfig().getOrderQuantity()))
            .setSide(bar.getClose().compareTo(bar.getOpen()) > 0 ? Side.BUY : Side.SELL)
            .build();

        getOrderService().sendOrder(order);
    }
}
```

The class EMAService method contains the following items:

1. Once the strategy has reached the START live cycle phase subscribe to the security needed for this strategy
2. Construct an Order Value Object using the MarketOrderVOBuilder. The OrderVO contains a reference to the strategy, the security, the account as well as the quantity and the order side.
3. Send the Order to the market via the OrderService

4.1.4.2. pom.xml

The Maven pom.xml file contains the Maven project definition as well as Maven dependencies:

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
4.1.4.3. Dockerfile

The Dockerfile contains all relevant information to build a Docker container:

```
FROM docker.algotrader.com/algotrader/algotrader:latest

ENV STRATEGY_NAME=EMA

WORKDIR /usr/local/strategy
ADD target/*.jar lib/

ENTRYPOINT ["/usr/local/algotrader/bin/docker-strategy-run.sh"]
```
4.1.5. Generated Artifacts Esper Archetype

The Esper Archetype will generate the following artifacts:

/src/main/java/ch/algotrader/strategy/EMAService.java
   The strategy service class

/src/main/resources/module-ema.epl
   Esper Module containing statements related to signal generation

/src/main/resources/conf-ema.properties
   Contains parameters used by the strategy (e.g. Moving average durations etc.)

/src/main/resources/META-INF/esper-ema.cfg.xml
   Contains event-types, imports, variables and general Esper settings

/src/main/resources/META-INF/applicationContext-client-ema.xml
   Spring Application Context File for the strategy

/src/main/resources/db/mysql/mysql-ema.sql
   MySQL data file containing db data needed for this trading strategy in live trading mode

/runConfigurations/*.xml
   IntelliJ IDEA (v.2020.1 or greater) Run Configurations to start the Strategy in embedded mode and simulation mode.

/pom.xml
   The Maven project object model file containing general information about the Trading Strategy

/Dockerfile
   The Docker file

4.1.5.1. EMAService.java

This is the main Java-class containing the Business Logic.

The references to the Services provided by the AlgoTrader Server (e.g. OrderService, PositionService, etc.) will be auto injected on startup by the Spring Framework

```java
public class EMAService extends StrategyService {
    private @Value("#{emaConfigParams.accountId}") long accountId;
    private @Value("#{emaConfigParams.securityId}") long securityId;
    private @Value("#{emaConfigParams.orderQuantity}") long orderQuantity;
```
@Override

public void onStart(final LifecycleEventVO event) {
    getSubscriptionService().subscribeMarketDataEvent(getStrategyName(), this.securityId, AdapterType.IB.name());
}

public void sendOrder(Side side) {

    MarketOrderVO order = MarketOrderVOBuilder.create()
            .setStrategyId(getStrategy().getId())
            .setAccountId(this.accountId)
            .setSecurityId(this.securityId)
            .setQuantity(this.orderQuantity)
            .setSide(side)
            .build();

    getOrderService().sendOrder(order);
}

The class EMAService method contains the following items:

1. Gets references to settings defined in conf-ema.properties
2. Once the strategy has reached the START live cycle phase subscribe to the security needed for this strategy
3. Construct an Order Value Object using the MarketOrderVOBuilder. The OrderVO contains a reference to the strategy, the security, the account as well as the quantity and the order side.
4. Send the Order to the market via the OrderService

4.1.5.2. module-ema.epl

This Esper module contains the Esper statements for signal generation

The statement MOVING_AVERAGE generate signals by using the moving average function.

the statement SEND_ORDER calls the EMAService.sendOrder() whenever there is a moving average crossover on the indicators. The Tag @Subscriber is used to instruct the EsperEngine to attach the Subscriber to this statement. This way the sendOrder method is called whenever there is a signal.
CONFIDENTIAL

Generated Artifacts Esper Archetype

TickVO;
@Name('SEND_ORDER')
@Subscriber(className='emaService#sendOrder')
select
case when indicator.value > 0 then Side.BUY else Side.SELL end as side
from
pattern [every indicator=Indicator]
where
(indicator.value > 0 and prior(1, indicator.value) <= 0)
or
(indicator.value < 0 and prior(1, indicator.value) >= 0)

4.1.5.3. conf-ema.properties
This configuration file contains the parameters of the strategy:
#{"type":"Integer","label":"Account ID"}
accountId = 1
#{"type":"Integer","label":"Security ID"}
securityId = 1
#{"type":"Integer","label":"Default Order Quantity"}
orderQuantity = 100000
#{"type":"Integer","label":"Moving Average Period Short"}
movingAveragePeriodShort = 10
#{"type":"Integer","label":"Moving Average Period Long"}
movingAveragePeriodLong = 20

4.1.5.4. esper-ema.cfg.xml
The esper configuration file looks like this:

<?xml version="1.0" encoding="UTF-8"?>
<esper-configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://espertech.com/schema/esper"
xsi:schemaLocation="http://espertech.com/schema/esper
http://espertech.com/schema/esper/esper-configuration-4-0.xsd">
<variable name="movingAveragePeriodShort" type="int" constant="true"/>
<variable name="movingAveragePeriodLong" type="int" constant="true"/>
</esper-configuration>

47


The above file configures the required variables along with their type. The actual values for the variables are taken from `conf-ema.properties`.

### 4.1.5.5. applicationContext-client-ema.xml

A typical Spring Configuration File looks like this:

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <!-- Bean Definitions -->
    <bean id="emaConfigParams"
          class="ch.algotrader.config.spring.CustomConfigParamsFactoryBean">
        <property name="global" ref="configParams"/>
        <property name="resource">
            <value>classpath:/conf-ema.properties</value>
        </property>
    </bean>

    <bean id="emaEngine"
          class="ch.algotrader.esper.EngineFactoryBean">
        <property name="strategyName" value="EMA"/>
        <property name="configResource" value="esper-ema.cfg.xml"/>
        <property name="configParams" ref="emaConfigParams"/>
        <property name="initModules" value="test"/>
    </bean>

    <bean id="emaService"
          class="ch.algotrader.EMAService" autowire="byName">
        <property name="strategyName" value="EMA"/>
        <property name="engine" ref="emaEngine"/>
    </bean>

</beans>
```

This file contains the following Spring Bean Definitions:

1. Contains a Map of all properties based on settings defined in `conf-ema.properties`.
2. Creates the Esper Engine based on `strategyName`, `configResource`, `configParams` and `initModules` and optional `runModules` definitions.
3. Creates the Strategy Service based on `strategyName` definition and `engine` reference. All dependencies of the `ch.algotrader.service.StrategyService` will be injected automatically through auto wiring.
4.1.5.6. mysql-ema.sql

The MySQL database script contains the following items:

```sql
INSERT INTO `strategy` (`ID`, `NAME`, `AUTO_ACTIVATE`, `VERSION`) VALUES
(2, 'EMA', True, 0);
```

The file contains an entry in the table strategy. The column `AUTO_ACTIVATE` means that the strategy will be automatically run in simulation mode.

4.1.5.7. pom.xml

The Maven pom.xml file contains the Maven project definition as well as Maven dependencies:

```xml
  <modelVersion>4.0.0</modelVersion>
  <groupId>algotrader</groupId>
  <artifactId>algotrader-ema</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ema strategy</name>

  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>algotrader</groupId>
      <artifactId>algotrader-core</artifactId>
      <version>...</version>
    </dependency>
  </dependencies>

  <properties>
    <java.version>1.8</java.version>
  </properties>
</project>
```
4.1.5.8. Dockerfile

The Dockerfile contains all relevant information to build a Docker container:

```sh
FROM docker.algotrader.com/algotrader/algotrader:latest

ENV STRATEGY_NAME=EMA

WORKDIR /usr/local/strategy
ADD target/*.jar lib/

ENTRYPOINT ["/usr/local/algotrader/bin/docker-strategy-run.sh"]
CMD ["-e"]
```

4.2. Building a Trading Strategy

Execute the following Maven command to start a Maven build of the trading strategy:

```sh
mvn install
```

The Maven modules can now be deployed to a Maven repository (e.g. Sonatype Nexus) using:

```sh
mvn deploy
```

For further details regarding a maven deploy please visit the [Maven deploy plug-in](https://maven.apache.org/plugins/maven-deploy-plugin/) page

Execute the following Docker command to create a Docker image of the trading strategy that can be used for productive deployments:

```sh
docker build -t xyz .
```

Please replace `xyz` with the name of the trading strategy.

The Docker image can now be pushed to a Docker repository (e.g. Docker Hub, Sonatype Nexus or Amazon ECR). For further details on pushing to a Docker repository please visit:

- [Docker Hub](https://docs.docker.com/docker-hub/repos/#pushing-a-repository-image-to-docker-hub)
- [Sonatype Nexus 3.0](https://support.sonatype.com/hc/en-us/articles/360000761828)

---

2. [https://docs.docker.com/docker-hub/repos/#pushing-a-repository-image-to-docker-hub](https://docs.docker.com/docker-hub/repos/#pushing-a-repository-image-to-docker-hub)
4.3. Hints for Strategy Development

It is possible to develop trading strategies purely in Java for simplicity. Very often it is though helpful to use Esper for Market Data Analysis and Signal Generation in addition to Java code. The following sections will provide hints on developing strategies both in Java and Esper.

4.3.1. Java based Strategies

Java bases Strategies typically consist of a single Java class where all logic is implemented inside event handler methods (e.g. `onInit`, `onBar`, `onTick`, `onOrderStatus`, etc.)

4.3.1.1. Strategy starters

The following two Starters are available to start a trading strategy in embedded and in distributed mode.

**StrategyStarter**

*StrategyStarter* starts a strategy in stand-alone mode running in a separate JVM process. The strategy will use ActiveMQ message broker to receive market data and other events from the server JVM process. The server JVM process is expected to be running before the strategy JVM is started.

**EmbeddedStrategyStarter**

*EmbeddedStrategyStarter* starts a strategy in single JVM (embedded) mode, when server and strategy run in the same process. Market data and other events are delivered directly to the strategy instances by a single event dispatcher.

4.3.1.2. Event Handler Methods

AlgoTrader is an event-based system. All strategy related events are propagated to strategies as event objects (e.g. `Order`, `OrderStatus`, `Tick`, `Bar`, etc.). Inside strategies these events are made available through event handler methods, e.g.:

```java
@override
public void onBar(BarVO bar) {
    // do something
}
```

4.3.1.3. Life-Cycle Events

Strategy classes can provide listeners for life-cycle events in order to receive notifications about strategy life-cycle phase transitions and execute custom life-cycle dependent logic.

---

4. Amazon ECR

https://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-push-ecr-image.html
For further information on life-cycle events please visit *Section 23.4, “Session life-cycle events”*

### 4.3.1.4. State based Strategy

Often a strategy has several states that it runs through during execution (e.g. FLAT, PENDING_LONG, PENDING_SHORT, LONG, SHORT, etc.). For these situations it is advisable to use a Java Enum, e.g.:

```java
package ch.algotrader.strategy;

public enum State {
    FLAT, PENDING_LONG, PENDING_SHORT, LONG, SHORT;
}
```

In case a strategy trades multiple instruments and each instrument has its own state it is suggested to create a `Metrics` object for each instrument. The `Metrics` object is a simple Java POJO that holds the state per instrument and potential other information regarding the instrument (e.g. values of technical indicators):

```java
public class Metrics implements Serializable {

    private static final long serialVersionUID = 5972079135237671512L;

    private long securityId;
    private State state;
    private double ema; // exponential moving average

    public Metrics(long securityId, State state, double ema) {
        this.securityId = securityId;
        this.state = state;
        this.ema = ema;
    }

    // getters and setters
```
4.3.1.5. Prevent an action from happening multiple times

Often a Java Action is triggered multiple times by a certain situation, because the underlying cause takes a finite amount of time to be resolved.

Example: The current market level exceeds a defined stop, which triggers a closing order. However during the time the order is being executed at the market, additional market data events are received. Because the position is not yet closed by that time, another undesired closing order might get placed.

To prevent actions from happening multiple times a state object mentioned in the previous section is again very helpful.

Whenever the predefined signal condition is met the state will be set to PENDING.

Below is an example implementation of a state based strategy.

```java
@Component
public class ABCService extends StrategyService {

    private final long[] securityIds = {1, 2, 3, 4, 5};

    private Map<Long, Metrics> metricsMap = new HashMap<>();

    @Override
    public void onInit(final LifecycleEventVO event) {
        for (long securityId : securityIds) {
            metricsMap.put(securityId, new Metrics(securityId, State.FLAT));
        }
    }

    @Override
    public void onTick(TickVO tick) {
        long securityId = tick.getSecurityId();
        Metrics metrics = metricsMap.get(securityId);

        if (metrics.getEma() > ... && State.FLAT == metrics.getState()) {
            metrics.setState(State.PENDING_LONG);
            sendOrder(securityId);
        }
    }

    private void sendOrder(long securityId) {
```
// create and send the order
}

@Override
public void onOrderStatus(OrderStatusVO orderStatus) {
    if (Status.EXECUTED == orderStatus.getStatus()) {
        Order order = getOrderService().getOrderByIntId(orderStatus.getIntId());
        Metrics metrics = metricsMap.get(order.getSecurity().getId());
        metrics.setState(State.LONG);
    }
}

1. List the securityIds the strategy wants to use
2. Store a Map of Metrics keyed by securityId
3. Create one Metrics object per securityId
4. Once a trigger was met (based on the trading logic) and the current State is FLAT set the State to PENDING
5. Send the order. The send order will not trigger again once the state has been set to PENDING_LONG.
6. Once the order got fully executed set the State to EXECUTED. As an alternative to using the onOrderStatus method a Section 8.4.8.2, “Trade callback” could be used.

4.3.1.6. Tagging of orders

Strategies that trade multiple securities at the same time often have the requirement to associate a particular order with a certain strategy. It is therefore often necessary to “tag” an order with additional meta data. This can be accomplished using order properties, see Section 17.2.3, “Order Properties”. In tagging orders using order properties it is also possible to distinguish automatically placed orders from manually placed orders (through AlgoTrader client or external broker GUI).

4.3.1.7. Using Base Strategy Names

Typically a strategy running corresponds to one entry in the database table strategy. In certain situations it may be necessary for a running strategy to place orders into separate entries in the database table strategy. Reasons for this might be:

- A strategy wants to have of multiple positions on the same instrument (i.e. long and short positions at the same time)
- A strategy needs to track positions for different accounts

For this purpose it is possible to use so called Base Strategy Names.

Example:
A strategy named EXAMPLE would like to interact with the two separate entries in the database named LONG and SHORT. For this purpose the strategy needs to be started with the strategyName set to EXAMPLE inside the file conf.properties. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```
strategyName = EXAMPLE
```

In addition one needs to create the two entries EXAMPLE|LONG and EXAMPLE|SHORT in the database table strategy. the | separator causes strategy events like OrderStatusVO, FillVO and TransactionVO are propagated to the running strategy EXAMPLE.

**Note**
Using other characters than | will not work and will cause strategy events not to arrive in the EXAMPLE strategy.

### 4.3.1.8. Rolling of Futures and Options

Due to the expiring nature of Futures and Options, corresponding Positions have to be rolled prior to the Expiration Date. Typically, this would involve the following steps:

1. Close the Front-Month Position
2. Unsubscribe the Front-Month Future/Option
3. Subscribe the Back-Month Future/Option
4. On First Tick (see Section 8.4.8.1, “First tick callback”) open a new Position

When dealing with futures one has to decide on when to roll from the Front-Month future into the Back-Month future. For this different philosophies exist:

- Roll on a fixed day prior to the expiration date or the first notice date
- Roll when the Back-Month future starts having a higher traded volume than the Front-Month future
- Use the constant maturity method as described in the following section

Since Futures have an expiration date and are therefore not continuous, it is often not possible to base indicators on them. There are several methods for dealing with this situation:

- use the raw data and ignore the fact that price time series will have jumps
- On the rollover day, add the difference between yesterday's closing price of the Back-Month future and yesterday's closing price of the Front-Month future to the combined time series. Alternatively one can use the difference between today's opening price of the Back-Month future and yesterday's closing price of the Front-Month future if only the time series for the generic 1st future is available. This method can be used for P&L calculation it might however lead to negative prices on long time series.
• Instead of using addition as mentioned in the previous item use multiplication. This method however cannot be used for P&L calculations. Depending on the indicator in use either addition or multiplication will be adequate.

• Use the constant maturity method as described in the following section

Please see Section 5.6, “Subscribing/Unsubscribing securities during a simulation” for options on how to back test futures and options based strategies that require multiple securities to be subscribed and unsubscribed during the back test.

4.3.1.9. Synchronizing system clock

Many strategies require that the local system clock is in sync with the remote server clock. Unfortunately it is not possible to directly sync the local time with the remote server clock. However most servers are using NTP or some other time sync mechanism to make sure there local clock is in sync with the official time defined by NTP servers. As a result local system clock should also be synchronized with NTP servers. In most cases this can be done directly through the operating system (e.g. Windows Time Service). For Windows Servers there is also the time sync tool available which tends to be more precise than the Windows Time Service.

4.3.1.10. Prevent LazyInitializationException

Strategy code running runs outside of Hibernate Sessions. Traversal along the Object Tree beyond what is already loaded into the Hibernate session will throw a LazyInitializationException. All n-to-1 associations (e.g. Position.getStrategy) will be fetched eagerly so no LazyInitializationException will be throws.

The LazyInitializationException can still occur on rare occasions. To handle those situations there is a corresponding method to fully initialize the association inside each Entity association (e.g. Transaction.initializeSecurity(Initializer initializer) or Combination.initializeComponents(Initializer initializer)). The method takes a reference to an Initializer. In strategies the CacheManager can be passed as an Initializer and in server-side services the HibernateInitializer can be used. Subsequent calls to the same association will then get access to the already initialized Entity or Collection.

Traverse of an uninitialized relation from Position to Security in strategy code would look like this:

```java
Position position = ... position.initializeSecurity(getCacheManager()); // initialize security position.getSecurity(); // security is now initialized
```

4.3.2. Esper based Strategies

When developing strategies using Esper in addition to Java code it is generally recommended to do Time-based Market Data Analysis and Signal Generation inside Esper Statements. Procedural actions like placing an order or subscribing to a Security are predominantly done inside Java Code.

5 http://www.timesynctool.com/
4.3.2.1. Print Statement Selects

By means of the TestSubscriber, selected values of a Statement can be printed to the Console.

```java
@Name('TEST')
@Subscriber(className='ch.algotrader.esper.subscriber.TestSubscriber')
@SimulationOnly
select
    valueA,
    valueB,
    valueC
from
    TestEvent;
```

4.3.2.2. Logging values of an Indicator to a log file

Often strategies are based on a technical indicator. During simulation it is often desirable to log the values of such an indicator to a log file. This can be done with the following statement:

```java
@Name('INSERT_INTO_INDICATOR')
@Listeners(classNames={'ch.algotrader.esper.listener.IndicatorListener'})
select
    dateTime.toDate() as dateTime,
    valueA,
    valueB
    valueC
from
    Indicator;
```

Above statement will log dateTime, valueA, valueB and valueC to the file files/report/IndicatorReport.csv

4.3.2.3. Access to Esper Variables

Esper has a sophisticated variable management functionality. It is possible to access those variables from Java through the following methods:

```java
// set variable value
engine.setVariableValue("target", target);

// retrieve variable value
Double target = (Double) engine.getVariableValue("target");
```
For further details on Esper Variables please visit the Esper Documentation\textsuperscript{6}

### 4.3.2.4. Esper Utility classes

Complex computations should be handled outside Esper. For this purpose, it is often easier to create a small Utility class and use its methods inside the statement. Example:

```java
package ch.algotrader.strategy;

public class MyUtil {

    public static double calculate(BigDecimal last) {

        return ...; // add calculation here
    }
}
```

This Utility class also needs to be declared in the file esper-...xml:

```xml
<auto-import import-name="ch.algotrader.strategy.MyUtil"/>
```

Now, the Utility class can be used in an Esper Statement like this to adjust a trailing stop loss:

```es
select
tick.last,
value
from
TickVO as tick,
method:MyUtil.calculate(tick.last) as value
```

For further details on using static methods please visit the Esper documentation\textsuperscript{7}

### 4.3.2.5. Prioritizing Statements

If two statements are based on the same Event, it is necessary to set a priority for each statement to make sure, the system behaves deterministically:

```es
@Name('STATEMENT_1')
@Priority(2)
select * from A;
```

\textsuperscript{6} http://esper.espertech.com/release-7.0.0/esper-reference/html/epl_clauses.html#variables_overview

\textsuperscript{7} http://esper.espertech.com/release-7.0.0/esper-reference/html/epl_clauses.html#joining_method_syntax
For further details on statement priorities please visit the Esper documentation

4.3.2.6. Market Data Event Pre-feeding

When starting up a strategy in Live Trading Mode, it is often necessary to initialize technical indicators that have a look-back period. This initialization is done by feeding historical market data into the Esper Engine.

A typical pre-feed method will look like this:

```java
@Override
protected void onPrefeed(final LifecycleEventVO event) {
    switch (event.getOperationMode()) {
        case REAL_TIME:
            feedMarketData();
            break;
    }
}

public void feedMarketData() {
    ZonedDateTime startDateTime = ZonedDateTime.now().minusHours(1);
    Collection<TickVO> ticks = getHistoricalDataService().getTicksByMinDate(securityId, startDateTime);
    DataFeedUtils.feedEventsToEngine(ticks, getEngine());
}
```

This method will load all ticks for the security (defined by `securityId`) for the last hour. It will then feed all of them sequentially to the local EsperEngine. Make sure collection you are providing to `feedEventsToEngine` method is sorted in ascending order by `dateTime`. In the example above `getTicksByMinDate` returns ticks in the correct order.

Note

Feeding of live market data only starts in the START live cycle phase. This prevents mix-up of historical pre-feed data and live market data.

4.3.2.7. State based Strategy

Often a strategy has several states that it runs through during execution (e.g. FLAT, PENDING_LONG, PENDING_SHORT, LONG, SHORT, etc.). For these situations it is advisable to use a Java Enum, e.g.:

---

```java
package ch.algotrader.strategy;

public enum State {
    FLAT, PENDING_LONG, PENDING_SHORT, LONG, SHORT;
}
```

In addition a corresponding Esper Variable has to be configured:

```xml
<variable name="state" type="ch.algotrader.strategy.State"/>
```

If the variable needs to have an initial value, the variable has to be declared with a statement.

```java
@Name('CREATE_VAR_STATE')
create variable ch.algotrader.enumeration.State state = State.FLAT;
```

It is now possible to query current state inside Esper statements like this:

```sql
select * from ... where state = State.FLAT;
```

In case a strategy trades multiple instruments and each instrument has its own state an Esper Named Window can be used instead of an Esper Variable.

```java
@Name('METRICS_WINDOW')
create window
    MetricsWindow.std:lastevent()
as
    Metrics;

@Name('INSERT_into_METRICS_WINDOW')
insert into
    MetricsWindow
select
    *
from
    Metrics;
```

The first statement creates the Named Window and the second statements inserts all Metrics events into the Named Window.

---

The Metrics object is a simple Java POJO that holds the state per instrument (and potential other information regarding the instrument):

```java
public class Metrics implements Serializable {

    private static final long serialVersionUID = 5972079135237671512L;

    private long securityId;
    private State state;

    public Metrics(long securityId, State state) {
        this.securityId = security;
        this.state = state;
    }

    // getters and setters
}
```

To initialize the Named Window one Metrics event per instrument has to be sent into the Esper Engine upon startup of the strategy:

```java
public void onInit(final LifecycleEventVO event) {
    getEngine().sendEvent(new Metrics(securityId, State.FLAT));
}
```

It is now possible to query current state inside Esper statements like this:

```sql
select state from MetricsWindow where securityId = ...;
```

### 4.3.2.8. Prevent an action from happening multiple times

Often a Java Action is triggered multiple times by a certain situation, because the underlying cause takes a finite amount of time to be resolved.

Example: The current market level exceeds a defined stop, which triggers a closing order. However during the time the order is being executed at the market, additional market data events are received. Because the position is not yet closed by that time, another undesired closing order might get placed.

To prevent actions from happening multiple times a state object mentioned in the previous section is again very helpful.

Whenever the predefined signal condition is met the state will be set to **PENDING**
Once the state has been changed to `PENDING_LONG` the statement `SEND_ORDER` will trigger an order to be sent. Since the `LONG_TRIGGER` statement only triggers if the state is either `SHORT` or `FLAT` it will not trigger again once the state has been set to `PENDING_LONG`.

Once the order has been fully executed (potentially using a Section 8.4.8.2, "Trade callback") the state needs to be changed to `LONG`.

```java
getEngine().executeQuery("update MetricsWindow set state = State.LONG where securityId = " + securityId);
```

### 4.3.2.9. Creation of Bars based on Ticks

Often strategies rely on indicators that are based on OHLC Bars. Since Live Market Data (i.e. Ticks) are not delivered in the format of Bars, it is often necessary to create Bars from arriving Ticks. Section 18.1, “Creation of Bars based on Ticks” explains how to do this.

### 4.3.2.10. Reacting upon a newly subscribed security

Especially for Option and Future based strategies it is often not possible to subscribe to the entire Option or Futures Chain in advance. Therefore the actual Security the strategy is interested in, is evaluated and subscribed to at runtime. There are often steps that should take place immediately after the first market data event has arrived for such a security.

```java
@Name('LONG_TRIGGER')
on
  //trigger event (can also be the MetricsWindow itself)
update
  MetricsWindow as metricsWindow
set
  state = State.PENDING_LONG
where
  // condition
and
  metricsWindow.state = State.SHORT or metricsWindow.state = State.FLAT

@Name('SEND_ORDER')
$Subscriber(className='xyzService#sendOrder')
select
  state
from
  MetricsWindow
where
  state = State.PENDING_LONG
or
  state = State.PENDING_SHORT;
```
Section 8.4.8.1, “First tick callback” explains how to use a FirstTickCallback for this purpose.

4.3.2.11. Reacting to an order execution

A common use case is to wait for the full execution or cancellation of an order and then take some additional action.

Section 8.4.8.2, “Trade callback” explains how to use a TradeCallback for this purpose.

In this context it is also important to remember that when trying to close a position there might still be open orders associated with the corresponding security and strategy. It is suggested to cancel all corresponding orders, attach a TradeCallback to the cancellation and only close the position once all cancels have been confirmed.

Also keep in mind that an order might receive multiple fills in live trading. For example if one wants to send a Stop Order for each executed Order it is important to use the filled quantity and not on the original order quantity.

In handling partial fills the TargetPositionOrder can be useful, please see: Chapter 24, Execution Algos

4.3.2.12. Waiting on market data session upon strategy startup

Upon system startup strategies run through the life cycle phases as defined in Section 3.2, “Live Trading Mode”. At the same time market data connections are established. Due to the asynchronous nature of these two processes it is not predetermined which one will complete first. Typically within the START life cycle phase market data is subscribed. This however will fail if the market data adapter has not reached its SUBSCRIBED state by that time. To circumvent this issue the following Esper pattern can be used, which waits for both the first LifecyclePhaseVO and SessionEventVO to arrive before it fires:

```
select *
from pattern[/LifecycleEventVO(phase=LifecyclePhase."START")
   and SessionEventVO(state=ConnectionState.SUBSCRIBED)];
```

4.3.2.13. Execute an action once a day at a certain time

```
@Subscriber(className='OrderService#cancelAllOrders')
select
   null
from
   pattern [every {timer:at(0, 18, *, *, 1:5)}];
```

The above statement will cancel all open orders at 18:00:00 Mo-Fri.
4.3.2.14. Prevent Memory leaks

All trading strategies allocate a certain amount of memory to objects. If however those object allocations are never released the corresponding memory will not get freed which will lead to a memory leak. This is especially a concern for strategies that are kept running for an extended period of time.

In addition one has to be careful with Esper statements not to introduce memory leaks. For example the following statement is potentially dangerous since it just keeps all Tick Events it receives:

```sql
select * from TickVO.win:keepall();
```

4.4. Strategy life-cycle events

The system provides life-cycle events to strategies when switching to another phase in the strategy life-cycle.

In addition the AlgoTrader life cycle manager supports two modes of operation: REALTIME and SIMULATION. In both modes all strategies transition through the same life-cycle phases. Depending on the operation mode not all phases may be relevant.

Table 4.1. Strategy life-cycle phase

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>Called after deploying all modules of the Server Engine but before deploying the init modules of the Strategy Engines.</td>
</tr>
<tr>
<td>PREFEED</td>
<td>Called after deploying the init modules with Engine#deployInitModules() of Strategy Engines but before deploying their run modules and before feeding any market data events.</td>
</tr>
<tr>
<td>START</td>
<td>Called after deploying the run modules of all Engines. At this time Market data events start feeding into strategy engines.</td>
</tr>
<tr>
<td>RUNNING</td>
<td>Called after the START phase to inform subscribers that all components have properly started and the system is fully up and ready to use.</td>
</tr>
<tr>
<td>EXIT</td>
<td>In SIMULATION mode this event occurs after finishing the simulation and before sending an EndOfSimulationVO event and before publishing simulation results. In REALTIME operation mode an EXIT life cycle event occurs when the virtual machine begins its shutdown.</td>
</tr>
</tbody>
</table>

Strategies can subscribe to these life-cycle events by overwriting the corresponding live-cycle method of the `StrategyService`:

```java
@override
public void onInit(final LifecycleEventVO event) {
  ...
}
```
public void onPrefeed(final LifecycleEventVO event) {
    ...
}

@Override
public void onStart(final LifecycleEventVO event) {
    ...
}

@Override
public void onRunning(final LifecycleEventVO event) {
    ...
}

@Override
public void onExit(final LifecycleEventVO event) {
    ...
}

Additionally, the user has the availability to subscribe to all incoming Life cycle events at once by overriding the onLifecycleEvent method instead. This is especially useful in Distributed mode, when there are two types of life-cycle: STRATEGY and SERVER.

@Override
public void onLifecycleEvent(final LifecycleEventVO event) {
    ...
}

Important

Overriding the onLifecycleEvents method directly is a non-standard approach and considered appropriate only in Distributed mode when the strategy should be aware of the Server JVM life-cycle as well. It requires the user to route different phase type events (INIT, PREFEED, START, EXIT) to appropriate methods and also differentiate those events by their type (SERVER vs STRATEGY). If this method is not overridden by the user, the SERVER events inside the Strategy JVM are ignored by default.

4.5. Strategy Development in Python

The AlgoTrader Python Interface provides a facility for implementation of strategies in Python 2 and 3.

For an example of a strategy written in Python see Appendix N, Example strategy “EMA” in Python and Appendix O, Example strategy “BreakOut” in Python
In order to test or run a Python based strategy, the strategy is implemented using AlgoTrader Python Interface classes and functions and runs in the Python interpreter. AlgoTrader is based on Java and needs to be run separately. Communication between Python and Java is implemented using the Py4J framework.

AlgoTrader (running in Java) can be started using the EmbeddedStrategyStarter (for live trading, or live trading against an exchange simulator, Section 5.1, “Exchange Simulator”), StrategyStarter (for the same purposes but in distributed mode, Section 3.2.2, “Distributed Mode”) or using the SimulationStarter for strategy backtesting on historical data (see Section 3.1, “Simulation Mode”). The Spring profile pythonIntegration needs to be activated with those.

In order to implement a strategy in Python, the main step is the same as in Java, extending the StrategyService class. It is found in the algotrader_com.services.strategy package.

The strategy implementation overrides event handler methods to receive data or events it needs from AlgoTrader. Method names mirror Java names in the Python code-style in lower case with underscores, e.g. on_bar is the equivalent of onBar in Java. Method parameters are identical.

Domain classes in Python mirror domain classes in Java, e.g. MarketOrder in Python is identical to MarketOrder in Java, except for names of fields changed to follow Python code-style. All classes including basic types are converted from their Java version to the Python one or vice versa automatically.

python_to_at_entry_point is present as a property in the parent StrategyService class to provide access to AlgoTrader services, e.g. OrderService for order placement via trading adapters, HistoricalDataService for retrieving historical data via historical data adapters, etc.

The following code sample shows how to place a market order on a new bar:

```python
class CustomStrategy(StrategyService):
    ...
    def on_bar(self, bar):
        market_order = MarketOrder(quantity=Decimal("10000"), side="BUY",
                                   strategy_id=1, account_id=101,
                                   security_id=25)
        self.python_to_at_entry_point.order_service.send_order(market_order)
    ...
```

The connect_to_algotrader function in the algotrader_com.interfaces.connection package is called to feed a custom strategy implementation (a class extending StrategyService class) into the AlgoTrader Python Interface.

The connect_to_algotrader function also has 3 optional parameters. First can be used to specify which event handler methods the strategy wants to subscribe to. This can be used as a means of performance optimization to prevent unnecessary calls from AlgoTrader to Python. If this optional parameter is not set, all event handler methods are subscribed automatically. Other two can be used to configure ports that will be used to communicate with AlgoTrader PythonService. It's needed to change default ports for every Python strategy running simultaneously. By default AlgoTrader uses ports 25333, 25334.
The `wait_for_algotrader_to_disconnect` function in the `algotrader_com.interfaces.connection` package is a convenience function to prevent Python scripts from finishing while the strategy is being run.

**Code example with both functions:**

```python
donly_subscribe_methods_list = ["onInit", "onStart", "onExit", "onBar", "onTick"]
_python_to_at_entry_point = 
connect_to_algotrader(EMAStrategyService(), only_subscribe_methods_list, 25335, 25336)
wait_for_algotrader_to_disconnect(_python_to_at_entry_point)
```

When overriding default ports in Python strategy it’s needed to change them also for Java part of the strategy. It can be done using following VM arguments in `StrategyStarter`:

```
-DpythonInterface.java.port=25335
-DpythonInterface.python.port=25336
```

Depending on the development environment you use you should have all necessary facilities for development, e.g. code completion functionality in PyCharm IDE:

```python
market_order = MarketOrder(quantity=ORDER_QUANTITY, side=side, strategy_id=strategy_id,
security_id=security_id)
python_to_at_entry_point.order_service.send_order(market_order)
self.first_order_sent = True
!previous_difference = difference
!ion_evolution.append(self.position)

_v = (data, window):
geometric moving average calculation
!window + 1.0)
1 - alpha
```

Quick documentation in PyCharm (ctrl+q):

```python
market_order = MarketOrder(quantity=ORDER_QUANTITY, side=side, strategy_id=strategy_id, account_id=account_id,
security_id=security_id)
python_to_at_entry_point.order_service.send_order(market_order)
```

AlgoTrader Python Interface uses type hint comments that are compatible with both 2 and 3 versions of Python. They serve as documentation and also enable static analysis of types used in the Python strategy code that uses the interface. Classes, methods and functions are documented using Google style docstrings.
Note that while services (e.g. OrderService) generally contain the same methods as their Java counterparts, some methods are not included and some have different names because Python doesn't allow method overloading (multiple methods with the same name and different parameters in the same class).

For a full documentation of available services and domain objects visit our Python Documentation\textsuperscript{10}

It is possible to use Section 5.4, “Automated Parameter Optimization” with Python based strategies. This is done by running AlgoTrader using SimulationStarter with optimizeSingleParamLinear, optimizeMultiParam, etc. options. Each iteration started by the optimization resets the strategy implementation to its initial state. Note that it’s necessary to write Python strategies so that they don’t use variables from outside of the strategy class scope. Variables not defined in the strategy class are not reset between optimization iterations and could wrongly affect the results of the simulation.

4.5.1. Python Strategy Performance

AlgoTrader is implemented in Java and the AlgoTrader Python Interface connects Python strategies with it using the Py4J framework. The interface uses synchronous calls with pinned threads to provide a reliable integration that also allows backtesting. While being fast, the integration adds a small latency to every call made between AlgoTrader and Python and vice versa. The latency for most calls averages around 100-200 microseconds, which is negligible for most strategies. Calls to some AlgoTrader services the AlgoTrader Python Interface provides however require several round trips. The method calls that are most heavily used have been optimized to only use one round trip. Note that the Python interpreter itself is slower than Java and that Python libraries are often not optimized for performance.

\textsuperscript{10} http://doc.algotrader.com/pythondoc/index.html
Strategy Backtesting

Historical data for back testing can be provided to strategies either via .csv files or via Section 19.1, “InfluxDB”.

Securities specified within the table subscription or securities subscribed to via the SubscriptionService are fed to the Strategy.

To feed data from CSV files during a back test the following property needs to be set inside conf.properties. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```
# should market data events be feed from CSV files
dataSource.0.dataSourceType=CSV
```

For further details on file format and storage location of CSV files please see Section 19.8, “Market Data File Format”.

**Note**

When feeding historical data with CSV files it is not possible to set a particular time range for the simulation. If this is a requirement please feed data through InfluxDB.

To feed data from InfluxDB during a back test the following properties need to be set inside conf.properties. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```
# should market data events be feed from the database
dataSource.0.dataSourceType=DB

# the backtest start date (ISO zoned date-time supported: 2016-01-01T08:30:30Z)
dataSource.0.minDate = 2016-01-01

# the backtest end date (ISO zoned date-time supported: 2016-12-31T22:00Z)
dataSource.0.maxDate = 2016-12-31
```

Backtesting strategies written in Python uses the same procedures as backtesting strategies written in Java, for details see Section 4.5, “Strategy Development in Python”

5.1. Exchange Simulator

The system provides an Exchange Simulator that is mainly used in back testing mode, but can also be used in live trading. The Exchange Simulator executes Orders by using an ExecutionModel. An Execution Model contains the logic which decides whether an order gets executed under the current market situation and what portion of the order gets executed. In addition the ExecutionModel also contains the logic to calculate commissions and fees that should be added to an order.

AlgoTrader contains a DefaultExecutionModel which provides a reasonable default logic for executing orders. Fees/commissions are calculated based on security/security_family properties:
• EXECUTION_COMMISSION_PER_ORDER
• EXECUTION_COMMISSION_PER_CONTRACT
• EXECUTION_COMMISSION_IN_PERCENT
• SLIPPAGE_PCT

The DefaultExecutionModel always charges commissions in the quote currency of the order. E.g. when buying BTC/USD the fees will be charged in USD.

For further details on the DefaultExecutionModel please consult the JavaDoc.

It is possible to replace the DefaultExecutionModel with a custom implementation that implements the interface ExecutionModel. The custom Execution Model needs to be registered as a Spring Bean in the following locations:

• For simulation: /META-INF/applicationContext-client-xxx.xml in the strategy project under src/main/resources.

• For live trading (to be used globally): /META-INF/applicationContext-env.xml. This file needs to be in the class path, e.g. in the conf project under src/main/resources.

During the simulation process transaction as well as position and cash_balance updates are executed in the database. It is therefore possible to use a standard database reporting tool to perform additional analysis on it.

To use the Exchange Simulator the Spring profile simulation has to be used, e.g.

-Dspring.profiles.active=simulation...

Note

Note than when using the Exchange Simulator in live trading, orders will be executed against live data internally and will not get sent to the external Broker or Exchange. If the Spring Profile simulation is enabled all other external Order Services will be disabled.

-Dsimulation=true denotes a back test and will effectively disable external Market Data services, so if the intention is to run exchange simulator against live market data, make sure that this parameter is set to false

All external order services must be disabled, e.g. Spring profiles like bFX, bFL etc must be inactive

5.2. Simulation Process

During a simulation process the following steps are executed sequentially by the SimulationExecutorImpl:
1. Create strategy entries in the database
2. The database is reset to its original state via the ResetService
3. An initial amount (USD 1'000'000 per default) is allocated to each strategy (the initial amount can be changed through the simulation.initialBalance setting inside conf.properties)
4. All server Esper modules are deployed
5. The life cycle phase INIT is broadcasted to all strategies. During this phase potential initiation steps can be invoked.
6. All strategy initModules Modules are deployed (if using Esper based strategies)
7. The life cycle phase PREFEED is broadcasted to all strategies. During this phase technical indicators can be initialized using historical data
8. All strategy runModules Modules are deployed (if using Esper based strategies)
9. Market data subscriptions are initialized based on entries in the table subscription
10. The life cycle phase START is broadcasted to all strategies. During this phase eventual actions like security subscriptions can be taken care of
11. At that time the actual simulation starts and market data events are starting to be sent into the Esper Engines
12. The life cycle phase EXIT is broadcasted to all strategies. During this phase eventual cleanup actions can be taken care of
13. At the end of each simulation run, metrics are printed to the console (if enabled), see Chapter 29, Metrics
14. All open orders are cancelled
15. All open positions are closed
16. An EndOfSimulationVO event is sent to all strategies
17. SimulationResults are retrieved from the strategies
18. Esper Engines are re-initialized
19. The Market Data Cache is flushed
20. The second-level cache is cleared
21. All reports are closed
22. The Excel based back test report is created and statistics are displayed to the console, see Section 5.5, “Performance Statistics”

5.3. Single Run Simulation

To run a strategy in Simulation Mode with the currently defined parameters use the procedure defined in Section 3.1, “Simulation Mode”.
5.4. Automated Parameter Optimization

The system allows running multiple simulations in parallel. Using cloud based servers thousands of simulation runs can be carried out in a matter of a few hours. For additional information please visit the full blog post on cloud based trading strategy optimization using AlgoTrader and Amazon Elastic MapReduce\(^1\).

Using Numerical Optimization functions (i.e. Brent & Newton) optimal parameter ranges can be determined in an automated fashion.

The following options exist (set in program arguments):

**simulateBySingleParam**

One Simulation run with a parameter set to the defined value. The example below will do one run with parameter a set to 0.8

```
simulateBySingleParam a:0.8
```

**simulateByMultiParam**

One Simulation run with multiple parameters set to defined values. The example below will do one run with parameter a set to 0.8 and b set to 12.0

```
simulateByMultiParam a:0.8, b:12.0
```

**optimizeSingleParamLinear**

Multiple Simulation runs by incrementing the value of one parameter within a defined interval. The example below will increment the value of parameter a starting at 0.1 to 0.9, incrementing by 0.1 for each run

```
optimizeSingleParamLinear a:0.1:0.9:0.1
```

**optimizeSingleParamByValues**

Multiple Simulation runs by iterating the value of one parameter according to defined list. The example below will iterate the value of parameter a through the following list: 0.2, 0.8, 0.9 and 1.2

```
optimizeSingleParamByValues a:0.2:0.8:0.9:1.2
```

**optimizeSingleParam**

Multiple Simulation runs by setting the value of one parameter within the defined range and trying to find the maximum Sharpe Ratio. The optimizer being used is UnivariateRealOptimizer. The example below will set the value of parameter a between 0.1 and 1.0 (accuracy 0.01).

Automated Parameter Optimization

**optimizeSingleParam**

`optimizeSingleParam a:0.1:1.0:0.01`

**optimizeMultiParamLinear**

Multiple Simulation runs by doing a matrix Optimization of 2 or 3 parameters by incrementing their values within a defined interval. The example below will iterate through all possible combinations by incrementing the value of parameter `a` starting at 0.1 to 0.9 (increment: 0.1), and incrementing the value of parameter `b` starting at 10.0 to 100.0 (increment: 5.0)

`optimizeMultiParamLinear a:0.1:0.9:0.1 b:10.0:100.0:5.0`

**optimizeMultiParam**

Multiple Simulation runs by adjusting the value of multiple parameters around their start values and trying to find the maximum Sharpe Ratio. The example below will start the optimization by setting the value of parameter `a` to 85.0 and parameter `b` to 150.0

`optimizeMultiParam a:85.0 b:150.0`

In order to process parameters with the correct decimal scale the following property needs to be updated inside `conf.properties`. Alternatively the property can be changed via Section 2.4, “VM Options”:

```
# the number of digits all portfolio balances will be displayed with
misc.portfolioDigits = 2
```

**Note**

In order for the parameter optimization to work the following properties need to be updated inside `conf.properties`. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```
# if set to true, writing to CSV reports will be disabled
report.disabled = true

# if set to true, the Excel back test report will open at the end of a simulation
report.openBackTestReport = true
```

**Note**

Before each back test run the Esper Engines will be reset. However Strategy services are not reset. Due to this any state that is maintained within the Strategy service needs to be reset within the `onInit` method.
Note

The values of Esper variables as well as Java properties get initialized on startup using Spring. The actual optimization only happens once the Spring context is fully initialized. Due to this it is necessary to overwrite the default values in the `onInit` from system.properties. This can be done as follows for Esper variables:

```java
getEngine().setVariableValue("propertyA", System.getProperty("propertyA"));
```

And like this for Java properties:

```java
this.propertyB = System.getProperty("propertyB");
```

5.5. Performance Statistics

At the end of each single simulation run, a CSV and Excel based back test report with performance statistics is created.
**Figure 5.1. Back Test Report**

<table>
<thead>
<tr>
<th>Draw Down</th>
<th>Worst Monthly Draw Down</th>
<th>1.03%</th>
<th>Best Monthly Performance</th>
<th>2.02%</th>
<th>Worst Draw Down (%)</th>
<th>2.36%</th>
<th>Worst Draw Down (Days)</th>
<th>46</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Monthly Average Return</td>
<td>0.79%</td>
<td>Monthly Standard Deviation</td>
<td>0.83%</td>
<td>Annual Average Return</td>
<td>9.77%</td>
<td>Annual Standard Deviation</td>
<td>3.23%</td>
</tr>
<tr>
<td></td>
<td>Count</td>
<td>428</td>
<td>%</td>
<td>54%</td>
<td>Total Profit</td>
<td>1277201.43</td>
<td>-1047487</td>
<td>229714.09</td>
</tr>
<tr>
<td></td>
<td>Average Profit</td>
<td>2'984.12</td>
<td>Average Profit (%)</td>
<td>0.23%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Losing</td>
<td>365</td>
<td>Losing</td>
<td>46%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>793</td>
<td>All</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>%</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>2.02</td>
<td>1.36</td>
<td>-0.10</td>
<td>1.46</td>
<td>-1.03</td>
<td>1.27</td>
<td>1.70</td>
<td>0.35</td>
<td>-0.31</td>
<td>1.96</td>
<td>1.25</td>
<td>0.78</td>
<td>11.20</td>
</tr>
<tr>
<td>2012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.29</td>
</tr>
<tr>
<td>2013</td>
<td>-0.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.29</td>
</tr>
</tbody>
</table>

**Performance Chart**

**Trades Chart**
The following 4 files are created in the sub-folder /files/report:

- **BackTestReport.xlsm**: the Excel based back test report (see image above)
- **MetricReport.csv**: contains key performance metrics
- **PortfolioReport.csv**: contains daily portfolio values (i.e. netLiqValue, marketValue, realizedPL, unrealizedPL, cashBalance, openPositions & leverage)
- **TradeReport.csv**: contains all trades including their profit

The Excel based back test report can be modified in terms of formatting and layout if needed.

In addition when running a single simulation run, statistics will be displayed to the console in the following format:

```
execution time (min): 2.43
dataSet: eurusd-1min-20111218-20130121
netLiqValue=1'229'714.00
month-year:          Dec-11  Jan-12  Feb-12  Mar-12  Apr-12  May-12  Jun-12  Jul-12
                  Aug-12  Sep-12  Oct-12  Nov-12  ... 
monthlyPerformance:   0.58%   4.03%   2.66% -0.19%   2.80%  -1.96%   2.44%   3.23%
                     0.66% -0.58%  3.67%  2.31%   ... 
year:                  2011    2012    2012
yearlyPerformance:    0.58%  22.33%  -0.06%
posMonths=10 negMonths=4 posYears=2 negYears=1
avgM=1.50% stdM=1.79% avgY=19.39% stdY=6.21% sharpeRatio=3.12
maxMonthlyDrawDown=1.96% bestMonthlyPerformance=4.03% maxDrawDown=4.49%
maxDrawDownPeriod=46.00days colmarRatio=4.32
WinningTrades: count=428(53.97%) totalProfit=1'277'201.37 avgProfit=2'984.12
               avgProfitPct=0.23%
LoosingTrades: count=365(46.03%) totalProfit=-1'047'487.34 avgProfit=-2'869.83
               avgProfitPct=-0.26%
AllTrades: count=793(100.00%) totalProfit=229'714.04 avgProfit=289.68
           avgProfitPct=0.00%
```

When running parameter optimizations, statistics will be displayed in the following summary format showing the current parameter values as well as corresponding performance statistics of one run on one single line:

```
a=90  avgY=39.86% stdY=20.16% sharpe=1.97 maxDDM=11.29% bestMP=8.35% ...
a=105 avgY=34.60% stdY=20.33% sharpe=1.69 maxDDM=11.56% bestMP=8.39% ...
```

In addition to above General Performance statistics, strategy specific performance statistics are printed to the console. These are retrieved by calling the method `StrategyService.getSimulationResults` of the strategy.

The amount of output during the simulation can be adjusted by setting the Log Level according to *Chapter 30, Logging*.
5.6. Subscribing/Unsubscribing securities during a simulation

By default, only those securities will be considered for simulations which have been subscribed to in the INIT or PREFEED phase.

Some strategies that are based on multiple securities need to subscribe and unsubscribe securities during the simulation. A typical example for this would be a Futures bases strategy that needs to unsubscribe an expiring Future and at the same time subscribe to the next Future in the chain.

As of now subscribing/unsubscribing in simulation is supported only for the InfluxDB event feed

```plaintext
dataSource.0.dataSourceType = DB
```

To subscribe or unsubscribe securities during a simulation call the respective method(s) of the SubscriptionService `Section 7.2.20, “Subscription Service”:

```plaintext
getSubscriptionService().unsubscribeMarketDataEvent(strategyName, securityId, adapterType);
getSubscriptionService().subscribeMarketDataEvent(strategyName, securityId, adapterType);
getSubscriptionService().unsubscribeFromOrderBook(strategyName, securityId, adapterType);
getSubscriptionService().subscribeToOrderBook(strategyName, securityId, adapterType);
```

Subscribing securities during strategy runtime adds some performance overhead, especially if you subscribe multiple securities at once. If you want to subscribe a batch of securities at once and increase performance, you can pause the data stream, add your subscriptions and then resume. This affects only simulation mode to make dataSource coordination more optimal. This has no effect on live mode and therefore can be safely used for both backtesting and live runs.

```plaintext
// pause simulation data sources
getMarketDataService().pauseSimulationData();
// subscribe multiple securities
getSubscriptionService().subscribeMarketDataEvent(...);
getSubscriptionService().subscribeMarketDataEvent(...);
// resume simulation data sources
getMarketDataService().resumeSimulationData();
```
Warning

As already mentioned, this feature is applicable only for the InfluxDB based simulation. When you subscribe/unsubscribe securities in a simulation, other data sources (e.g. CSV) will be reset and closed, and will stop supplying events until the end of the simulation or throw an Exception.
Architecture

The architecture of AlgoTrader is composed of the following components.

**Figure 6.1. Architecture**

The AlgoTrader Server provides the infrastructure for all strategies running on top of it. The AlgoTrader Server holds the main Esper Complex Event Processing (CEP) engine. It is responsible for all domain model objects and their persistence in the database. Different market data adapters are available to process live and historical market data. On the other end adapters for different execution brokers and exchanges are available, which are responsible for placing orders and receiving executions.

The AlgoTrader Server also provides business components for back testing, parameter optimization, analysis, execution management, risk management, reporting and hedging.

On top of the AlgoTrader Server any number of strategies can be deployed. Strategies can either be coded purely in Java or in a combination of Java and Esper code. Esper based strategies make use of a dedicated Esper CEP engine. A strategy can deploy any number of SQL-like Esper statements for time-based market data analysis and signals generation. Esper statements can invoke any number of procedural actions, such as placing an order or closing a position, which are coded in Java. The combination of Esper statements and Java Code provides a best-of-both-worlds approach.

For management and monitoring of the system different GUI clients exist. The AlgoTrader UI provides trading related functionality like charting, orders, positions & market data. In this manual we describe using IntelliJ IDEA for strategy development.

For productive installations and deployment AlgoTrader uses Docker.
Domain Model

The following sections describe the Domain Model of the system using UML (unified modeling language).

7.1. Entities

Figure 7.1. Entities Overview

The Main Entities of the system are specified within the following table:

Table 7.1. Entities

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>Each object of this class represents a running strategy within the system</td>
</tr>
<tr>
<td>Security</td>
<td>This is the base class of all securities in the system</td>
</tr>
<tr>
<td>SecurityFamily</td>
<td>A group of Securities (e.g. all S&amp;P 500 Futures)</td>
</tr>
<tr>
<td>Subscription</td>
<td>Market Data Subscriptions of a Strategy for a particular Security are represented by this class. For every Subscription the Strategy will receive Live Market Data for the corresponding Security</td>
</tr>
<tr>
<td>MarketDataEvent</td>
<td>Represents any type of market data related to a particular Security</td>
</tr>
<tr>
<td>Order</td>
<td>An Order for a particular Security</td>
</tr>
<tr>
<td>Account</td>
<td>An account held with an external Broker or Exchange</td>
</tr>
</tbody>
</table>
A full list of all Entities of the system will be discussed throughout the next sections. Entities of the system can be categorized into the following three areas:

**Reference Data**

Represent static referential data like:


**Market Data**

Represent external events (Tick and Bar) coming from market data providers or internal events (Generic Events) coming from another trading strategy. Market Data is typically immutable and of only momentary interest to the trading strategies. Market Data Events are available as Value Objects only (but not as Entities):

- MarketDataEventVO and its subclasses TickVO, BarVO, QuoteVO, BidVO, AskVO, TradeVO and GenericTickVO as well as any type of GenericEventVO

**Transaction Data**

Represent the financial state of trading strategies. Some of them (e.g. Transactions and Measurements) are immutable whereas others (e.g. Positions and Balances) are mutable and change their values while Orders are getting executed:

- Order, Transaction, Position, CashBalance, Measurement, PortfolioValue and related Entities

Besides providing Getters and Setters all Entities provide the following common features:

**VO Converter**

The static inner Converter class can be used to automatically convert the Entity to its corresponding Value Object, see Section 7.3, “Value Object”

**Factory**

The static inner Factory class can be used to create new instances of an Entity
7.1.1. Strategy

The strategy entity represents an individual strategy running inside AlgoTrader.

Regarding the question "what is a productive strategy?". It essentially up to the user, what he would like to consider as one strategy. A strategy can have one or multiple instruments. And also regarding trading logic there is no limitation.

However please note that the entire performance and reporting functionality of AlgoTrader happens on the strategy level. So if one would like to see performance metrics on an instrument level one would have to instantiate multiple strategies. Also, if it is a requirement to start and stop individual functions separately, it is best to put them into two separate strategies.

On the technical side each separate strategy allocates a certain amount of overhead (memory and CPU). For that reason it is best to combine functionality into as few strategies as possible if there are no good reasons not to separate them.

The field autoActivate means that if a strategy is set to active corresponding market data subscriptions are initiated automatically upon startup of the system. This is useful in distributed mode when strategies and the server run in different processes. If you restart the server in this scenario, subscriptions for the strategies are automatically loaded again (without having to restart the strategies).
There are several classes that are directly related to the strategy.

### Table 7.2. Strategy Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortfolioValue</td>
<td>On regular time intervals certain portfolio values (e.g. NetLiqValue, CashBalance, etc.) are saved to the database for every strategy.</td>
</tr>
<tr>
<td>Measurement</td>
<td>Custom Measurements (e.g. current value of a custom indicator) related to a strategy can be saved using this class</td>
</tr>
<tr>
<td>CashBalance</td>
<td>A <code>CashBalance</code> represents the current cash amount of a particular strategy in a particular currency</td>
</tr>
</tbody>
</table>

### Table 7.3. Portfolio Value Details

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cashBalance</td>
<td>Market value of all open forex positions + cash amount available to the strategy</td>
</tr>
<tr>
<td>marketValue</td>
<td>Market value of all open (non-forex) positions</td>
</tr>
<tr>
<td>netLiqValue</td>
<td>Cash balance + market value</td>
</tr>
<tr>
<td>realizedPL</td>
<td>Realized P&amp;L of all positions</td>
</tr>
<tr>
<td>unrealizedPL</td>
<td>Unrealized P&amp;L of all positions</td>
</tr>
</tbody>
</table>

All valuations (strategy and position level) can be queried via the `PortfolioService`. Fees are considered in the calculations if properly configured.
7.1.2. Security

Figure 7.3. Securities

The above UML Class diagram shows all available Security classes

Table 7.4. Security Types

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>A tradable Option</td>
</tr>
<tr>
<td>Future</td>
<td>A tradable Future</td>
</tr>
<tr>
<td>Forex</td>
<td>A Foreign Exchange Currency (FX) or Crypto Currency</td>
</tr>
<tr>
<td>Stock</td>
<td>A Single Stock</td>
</tr>
<tr>
<td>Fund</td>
<td>An ETF, Mutual Fund, etc.</td>
</tr>
<tr>
<td>Index</td>
<td>An Index (e.g. Equity, Volatility, Commodity)</td>
</tr>
<tr>
<td>GenericFuture</td>
<td>A virtual Future with a fixed duration</td>
</tr>
</tbody>
</table>
## Market Data Events

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InterestRate</td>
<td>Any type of Interest Rate</td>
</tr>
<tr>
<td>Bond</td>
<td>A corporate or government Bond</td>
</tr>
<tr>
<td>Commodity</td>
<td>A physical Commodity (e.g. Energy, Metals, Agriculture or Livestock). For Commodity Futures use Future.</td>
</tr>
<tr>
<td>PerpetualSwap</td>
<td>A perpetual swap contract supported by various crypto derivatives exchanges (e.g. BitMEX)</td>
</tr>
<tr>
<td>Combination</td>
<td>A synthetic security composed of one or many Components (see Chapter 25, Synthetic Securities and Derivative Spreads)</td>
</tr>
<tr>
<td>SecurityReference</td>
<td>A generic link between one security and another</td>
</tr>
</tbody>
</table>

The `Security` class provides the following two important methods:

- `getValue` which calculates the current (market) value of the instrument based on a quantity and a price parameter
- `getPrice` which calculates the current price of the instrument based on a (market) value and a quantity parameter

For most instruments the formula is:

- \[ \text{value} = \text{quantity} \times \text{contractSize} \times \text{price} \]
- \[ \text{price} = \frac{\text{value}}{\text{quantity}} / \text{contractSize} \]

However for `PerpetualSwaps` the formulas are different:

- \[ \text{value} = \text{quantity} \times \text{contractSize} / \text{price} \]
- \[ \text{price} = \text{quantity} \times \text{contractSize} / \text{value} \]

A Security Family contains common information about an entire family of securities (i.e. all general information about options on S&P500 are stored using this class). The class provides fields like `MIN_QTY`, `MAX_QTY` and `MIN_PRICE`. This information is used by `ReferenceDataService` when downloading new future and option changes, values from Security Family will then be copied onto the newly created Futures and/or Options. In regular operation mode (i.e. simulation of live trading) the information from Security Families are not used but only the information contained within Securities.

`SecurityReference` is a generic link between one security the owner and another the target. Using this class it is possible for a Security to have links to multiple other Securities.

### 7.1.3. Market Data Events

Market Data Events are available as Value Objects only but not as Entities. There are three different kinds of Market Data Events:
### Table 7.5. Market Data Types

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarVO</td>
<td>Open-High-Low-Close Price Bars, also containing volumes and volume weighted average prices</td>
</tr>
<tr>
<td>TickVO</td>
<td>Snapshot of the market at a particular point in time, containing information like last price, last time, bid, ask, volume, etc.</td>
</tr>
<tr>
<td>QuoteVO</td>
<td>Its subclasses represent the current best bid and offer: BidVO and AskVO</td>
</tr>
<tr>
<td>TradeVO</td>
<td>An actual order that was executed on the market, containing information like last price, last size and volume</td>
</tr>
<tr>
<td>GenericTickVO</td>
<td>Represents additional price information made available by market data providers (e.g. open price, close price, vwap price)</td>
</tr>
<tr>
<td>OrderBookVO</td>
<td>A snapshot of the order book for a security from the trading venue at a particular moment in time. It contains bid and ask price levels represented by the class OrderBookLevelVO. OrderBookLevelVO, in turn, contains a price, the number of orders with that price and the total amount of a security in those orders.</td>
</tr>
<tr>
<td>AggregatedOrderBookVO</td>
<td>Combines by symbol order books for a security from multiple venues. Like its regular counterpart, it contains bid and ask price levels represented by the class AggregatedOrderBookLevelVO. The difference between AggregatedOrderBookLevelVO and OrderBookLevelVO is that the former provides the number of orders and the amount of a security summarized across venues.</td>
</tr>
</tbody>
</table>

*aCumulative traded volume within the specified interval, stored in the tradedVolumeInInterval field of TickVO. The interval length in seconds is defined by the tradedVolumeInterval configuration property.

For simulation purposes Bars and Ticks can be supplied through CSV files (see Section 19.8, “Market Data File Format”) or through InfluxDB (see Chapter 19, Historical Data). In live trading Trades, Bids and Asks are received by the broker / exchange specific MarketDataService.

For conversion between Ticks and Bars please see Section 18.1, “Creation of Bars based on Ticks”.

---

**Note**

There are two properties associated with OrderBookVO:

- `misc.maxOrderBookVOdepth` sets the max distance from mid-price in an order book, effectively reducing the number of order book levels available.
- `misc.minNumOrderBookLevels` sets the minimal number of levels on each side of the order book in case the `misc.maxOrderBookVOdepth` filter was too strict and caused removing a lot of levels (applies often for low-priced securities).
• `{adapter}.orderbookfilter` sets the minimum price level for an order book, where `{adapter}` is a shorthand for adapter, e.g. `bmx.orderbookfilter` for Bitmex adapter

---

**Note**

The `AggregatedOrderBookVO` has four filtering methods:

• `getCumulatedBids(BigDecimal cumulationLevel)` sums up bid levels with price below the one given in `cumulationLevel` into a single level

• `getCumulatedBids(BigDecimal cumulationLevel, List<Long> exchangeIds)` selects bid levels with the `exchangeIds` before summing them up as in `getCumulatedBids(BigDecimal cumulationLevel)`

• `getCumulatedAsks(BigDecimal cumulationLevel)` sums up ask levels with price above the one given in `cumulationLevel` into a single level

• `getCumulatedAsks(BigDecimal cumulationLevel, List<Long> exchangeIds)` selects ask levels with the `exchangeIds` before summing them up as in `getCumulatedAsks(BigDecimal cumulationLevel)`
7.1.4. Order

The following UML Class diagram shows the Order and its related subclasses.

Table 7.6. Order Classes

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Base Class for all Order Types</td>
</tr>
<tr>
<td>OrderStatus</td>
<td>Order Status changes received back from the Broker / Exchange (e.g. PARTIALLY_EXECUTED or CANCELLED) are represented by this class</td>
</tr>
<tr>
<td>OrderCompletion</td>
<td>Similar to Order Status but only gets created once an order is fully executed or cancelled and all corresponding database activity has been completed.</td>
</tr>
<tr>
<td>OrderProperty</td>
<td>An arbitrary property that can be attached to an Order. Through the type attribute the OrderProperty can be marked as internal only or as IB property.</td>
</tr>
<tr>
<td>Fill</td>
<td>Filled orders are represented by this Class</td>
</tr>
<tr>
<td>Transaction</td>
<td>Each Fill is recorded as a transaction in the database using this entity. In addition the table transaction also carries transactions like INTREST, DEBIT, CREDIT &amp; FEES</td>
</tr>
<tr>
<td>Entity</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SimpleOrder</td>
<td>An Order that can be sent directly to the market</td>
</tr>
<tr>
<td>MarketOrder</td>
<td>Predefined SimpleOrder types</td>
</tr>
<tr>
<td>LimitOrder</td>
<td></td>
</tr>
<tr>
<td>StopOrder</td>
<td></td>
</tr>
<tr>
<td>StopLimitOrder</td>
<td></td>
</tr>
<tr>
<td>AlgoOrder</td>
<td>A composite order that will generate multiple SimpleOrders. An AlgoOrder cannot be sent directly to the market. AlgoOrders are also called “Execution Algos”, see Chapter 24, Execution Algos</td>
</tr>
<tr>
<td>TWAPOrder</td>
<td>This algorithm aims to match the Time-Weighted Average Price</td>
</tr>
<tr>
<td>VWAPOrder</td>
<td>This algorithm aims to match the Volume-Weighted Average Price</td>
</tr>
<tr>
<td>TargetPositionOrder</td>
<td>This algorithm automatically manages orders to reach the specified target quantity.</td>
</tr>
<tr>
<td>TrailingLimitOrder</td>
<td>This algorithm submits an order directly to the broker/exchange, with a limit price set a fixed distance from the current market price.</td>
</tr>
<tr>
<td>SlicingOrder</td>
<td>An AlgoOrder, that will split a large order into multiple child orders. The size of the child order, time in the market and delay between orders are randomized within the specified range.</td>
</tr>
</tbody>
</table>

**Note**

AlgoOrders and Order parent/child associations are persisted to the database. After a system restart, pending AlgoOrder will be visible but it will not continue execution automatically - it will not create new child orders. Execution reports for existing child orders will be processed.

### 7.1.5. Account

```java
<<Entity>>

Account
+ name : String(algoTrader_property_use_in_equals)
+ active : boolean(nonunique)
+ adapterType : String(nonunique)
+ accountServiceType : String [0..1](nonunique)
+ accountQualifier : String [0..1](nonunique)
+ alAccount : String [0..1]
+ alAccountGroup : String [0..1]
+ alAllocationProfile : String [0..1]
+ alClearingAccount : String [0..1]
+ alCngid : Long [0..1]
```

**Figure 7.5. Account**
An Account represents either an actual account, an account group (IB specific) or an allocation profile (IB specific). An account is assigned to a particular adapterType (e.g. IB_NATIVE or FXCM_FIX) which identifies the OrderService to use for this account. In addition the field sessionQualifier which is needed to define the actual session in place (for FIX Connections). With this setup, it is possible to have multiple Sessions (session qualifiers) per AdapterType and to have multiple Accounts per Session. If the field active is set to true a potential corresponding Fix session will be activated.

Optionally an accountServiceType (e.g. IB_NATIVE or BFX) can be added which identifies the AccountService to use for this account.

Accounts have an optional dependency to Exchange for cases when an account can only be used to trade on one single Exchange (typical for Crypto Currency Exchanges).

**Note**

Orders sent to the market will always contain Account related information in an adequate way (e.g. as a FIX Tag 1). Also Transactions which are based on an actual order will have an association with a particular Account. However Positions do not hold any information regarding Accounts. It is thus possible that a Position holds aggregated Quantities from several external Accounts. Also it is possible to open a position through on account but then close it through another (i.e. when using separate execution and clearing brokers). With this setup Strategies do not have to worry about the actual Accounts the funds are located in. This way, a strategy will always only see one Position per Security.
Each Fill is recorded as a transaction in the database using this entity. In addition the table transaction also stores transactions like \textsc{interest}, \textsc{debit}, \textsc{credit} & \textsc{fees}. A transaction is immutable and contains all relevant information like date\textit{time}, quantity, price, execution\textit{Commissions}, clearing\textit{Commissions} and fees as well as references to Account, Strategy, Security and Position.

Depending on the type of transaction the field \textit{quantity} has the following values:

- **BUY**: > 0
- **SELL**: < 0
- **EXPIRATION**: any value
- **TRANSFER**: any value
- **CREDIT**: 1
- **EXCHANGE\_CREDIT**: 1
- **INTEREST\_RECEIVED**: 1
• REFUND : 1
• DIVIDEND : 1
• DEBIT: -1
• EXCHANGE_DEBIT: -1
• INTREST_PAID: -1
• FEES: -1 (+1 for maker FEES paid)

As the sign of the transaction is defined by the quantity the fields executionCommissions, clearingCommissions and fees will always be positive for fees/commissions charged (they will be negative for fees/commissions paid, e.g. maker rebates).

Some crypto exchanges provide fee information. If the fees are in the currency of the transaction, they are stored in the fee attribute of the transaction. In case the fees are charged in another currency (for example Binance charges in its own currency - BNB), a new transaction is created with transactionType = FEES, quantity = -1 for fees paid (and quantity = -1 for fees received, e.g. maker rebates), price = fee value and currency = fee currency.

7.1.7. Position

For any Strategy holding a particular Security a Position is created in the database. Even if this position is later on closed (i.e. quantity = 0) the position will still remain in the database, because the associated Transactions still have references to it.

In general, position values (e.g. marketPrice, marketValue, averagePrice, cost, unrealizedPL & realizedPL) are calculated per actual strategy related position and show the price that would need to payed if the position was closed at this moment.
Table 7.7. Position Valuation Details

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>realizedPL</td>
<td>Total profit of closed parts of a position (parts of a position might still be open)</td>
</tr>
<tr>
<td>unrealizedPL</td>
<td>Profit of the currently open part of a position</td>
</tr>
<tr>
<td>cost</td>
<td>Total cost incurred to open the current position (potentially through multiple orders). These values are based on the fee configurations</td>
</tr>
</tbody>
</table>

All valuations (strategy and position level) are available through the Section 7.2.12, “Portfolio Service”.

7.1.8. Cash Balance

A CashBalance represents the current cash amount of a particular strategy in a particular currency.

Warning

Cash Balances are derived by taking all Transactions of the given Security and Strategy into account. It is therefore important not to modify Cash Balance entries directly in the database. In case transactions are added or modified manually to the database, please use the management action reset position and cash balances in the Figure 11.3, “AlgoTrader UI Settings”

7.1.9. Subscription

Figure 7.8. Subscription

Market Data Subscriptions of a Strategy for particular Securities are represented by this class. For every Subscription the Strategy will receive Live Market Data for the corresponding Security
7.1.10. Exchange

Exchanges around the world have different trading hours. Quite often there are different trading hours even for different securities trading on the same exchange. In addition each exchange typically has different holidays or days where trading starts late or trading stops early. Especially for futures trading there are often small gaps between different trading periods of the same trading date. FX trading is often available 24 hours a day without any gaps.

All of these scenarios are captured and maintained through the Entities Exchange, TradingHours and Holiday:

Table 7.8. Exchange

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exchange</td>
<td>Represents an individual Security, a group of Securities or an entire Exchange (if all Securities have the same trading hours). An Exchange has a name, a code (typically MIC) as well as a time zone.</td>
</tr>
<tr>
<td>TradingHours</td>
<td>Defines an individual trading period (e.g. 09:00am to 16:30pm). In addition TradingHours identify the weekdays they are valid for.</td>
</tr>
<tr>
<td>Holiday</td>
<td>Identifies a holiday of a specific exchange. In addition a Holiday can identify a late opening or early closing of trading on a particular trading day.</td>
</tr>
</tbody>
</table>
7.1.11. Order Preference

### Table 7.9. Order Preference

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderPreference</td>
<td>This class allows definition of order default values (e.g. account, order type, delays, etc.). Except for the order type, all values have to be defined through Properties.</td>
</tr>
</tbody>
</table>

for further details see Section 17.2.1, “Order Preferences”
7.1.12. Quote Request and Quote

A request for quote (RFQ) is a process in which a company requests a quote from broker/exchange for the purchase or sale of a specific security.

Table 7.10. Quote Request and Quote

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quote_Request</td>
<td>This class allows definition of request sent to broker/exchange for the purchase or sale of specific security.</td>
</tr>
<tr>
<td>Quote</td>
<td>This class represents response provided by broker/exchange for the quote request.</td>
</tr>
</tbody>
</table>
7.2. Services

The system is based on a Service Oriented Architecture (SOA). All operations of the system are provided as Spring Services / Beans. The following groups of services exist:

1. Main Services, are available to both the AlgoTrader Server and Strategies
2. Client Services, which will be instantiated by each Strategy (and the AlgoTrader Server itself)
3. Private Services, which are only used by the AlgoTrader Server

For a full list of all Services please visit our JavaDoc\(^1\)

Inside strategies all services are injected by the Spring Framework and can be accessed as follows withing the strategy service:

```java
// subscribe for live market data
getSubscriptionService().subscribeMarketDataEvent(strategyName, securityId, adapterType);

// lookup a instrument by symbol
getLookupService().getSecurityBySymbol(symbol);

// send an order to the broker or exchange
getOrderService().sendOrder(order);
```

7.2.1. Main Services

Table 7.11. Main Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountService</td>
<td>Responsible for retrieval of account balances and initiation of withdrawals</td>
</tr>
<tr>
<td>CalendarService</td>
<td>Responsible for information about Exchange trading hours and holidays</td>
</tr>
<tr>
<td>CombinationService</td>
<td>Responsible for handling all Combination / Component related DB-Operations.</td>
</tr>
<tr>
<td>FutureService</td>
<td>Responsible for all future specific operations</td>
</tr>
<tr>
<td>HistoricalDataService</td>
<td>Responsible for the retrieval of historical data from Historical Data Providers</td>
</tr>
<tr>
<td>MarketDataService</td>
<td>Responsible for the retrieval of market data as well as Subscription Management.</td>
</tr>
</tbody>
</table>

\(^1\) http://doc.algotrader.com/javadoc/ch/algotrader/service/package-frame.html
### Service Description

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeasurementService</td>
<td>Responsible for persistence and retrieval of Measurements related to Strategy</td>
</tr>
<tr>
<td>OptionService</td>
<td>Responsible for all option specific operations</td>
</tr>
<tr>
<td>OrderService</td>
<td>Responsible for sending orders to the Broker / Exchange</td>
</tr>
<tr>
<td>PortfolioService</td>
<td>Responsible for providing portfolio values</td>
</tr>
<tr>
<td>PositionService</td>
<td>Responsible for management of positions, e.g. close position and reduce position</td>
</tr>
<tr>
<td>PropertyService</td>
<td>Responsible for persistence of Properties related to a PropertyHolder</td>
</tr>
<tr>
<td>ReferenceDataService</td>
<td>Responsible for the download of option and future chains</td>
</tr>
<tr>
<td>RfqService</td>
<td>Responsible for sending requests for quotes to the Broker / Exchange</td>
</tr>
</tbody>
</table>

7.2.2. Client Services

Table 7.12. Client Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarketDataCacheService</td>
<td>Provides a strategy local cache of market data and FX conversion rates</td>
</tr>
<tr>
<td>LookupService</td>
<td>Provides general data lookup operations to other services</td>
</tr>
<tr>
<td>ConfigAwareStrategyService</td>
<td>Base class for all strategy services which has references to all necessary services and implements all event listener interfaces. In addition the service receives a reference to the strategy config</td>
</tr>
<tr>
<td>StrategyService</td>
<td>Base class for all strategy services which has references to all necessary services and implements all event listener interfaces</td>
</tr>
<tr>
<td>SubscriptionService</td>
<td>This service is used by the strategy for subscription management. The actual DB related operations are carried out by the MarketDataService. The MarketDataService should not be called directly by strategies.</td>
</tr>
</tbody>
</table>

7.2.3. Account Service

The AccountService interface defines a method for retrieving account balances as well as the initiation of crypto withdrawals for crypto exchanges. For further details see Chapter 21, Account Data.

7.2.4. Calendar Service

The CalendarService is responsible for providing information about Exchange trading hours and holidays.

Especially when trading multiple exchanges around the globe the CalendarService becomes very useful. It provides convenient methods like:

- `isOpen` (is the specified exchange open at the current time or at the specified date time). Will return true if no TradingHours are defined.
• `isTradingDay` (is the current day or the specified day a trading day at the specified exchange)

• `getOpenTime` (gets the open time of the specified exchange on the current day or the specified day)

• `getCloseTime` (gets the close time of the specified exchange on the current day or the specified day)

• `getNextOpenTime` (gets the next open time of the specified exchange after the current date time or the specified date time)

• `getNextCloseTime` (gets the close open time of the specified exchange after the current date time or the specified date time)

In addition the Calendar service provides methods to identify a particular trading day, which will be important to associate a particular order for clearing. If a trading session overlaps from one day to another (e.g. starts on Sunday 23:00pm), the trading day will be considered the day when the session ends (e.g. Monday). However in this example Monday would need to be set to `true` in the corresponding `TradingHours` object.

**Note**

All dates and times in the `CalendarService` are converted to the system time, e.g. the market opens at 09:30 EST but the system time zone is CET then the market opening time in the `CalendarService` will be 15:30.

When trading one single exchange it is usually easiest to set the system time to the same time zone of the exchange.

When trading exchanges in different time zones one has the choice of setting the system to clock to the same time zone as one of the exchanges or leave the system time set to the local time zone.

### 7.2.5. Combination Service

AlgoTrader supports Synthetic Securities & Derivative Spreads. A Combination consists of one or many Components. For further details see *Chapter 25, Synthetic Securities and Derivative Spreads*.

### 7.2.6. Future Service

AlgoTrader has full support for Future based trading strategies. For further details see *Chapter 15, Options & Futures*

### 7.2.7. Historical Data Service

AlgoTrader provides several Historical Data Interfaces out-of-the-box. The system can store historical data in the integrated *Section 19.1, “InfluxDB”* and feed stored or recorded historical data to strategies during back tests. The system also integrates a feature for live data recording as well as live tick-to-bar aggregation. For further details please see *Chapter 19, Historical Data*
7.2.8. Market Data Service

AlgoTrader provides several Market Data Interfaces out-of-the-box. Live market data is available to trading strategies running within the system. For further details please see *Chapter 18, Market Data*

7.2.9. Measurement Service

The *MeasurementService* allows storage of arbitrary measurements in the database. Measurements contain a name, a time stamp and a value of type `Integer`, `Double`, `Money` (`BigDecimal`), `Text` (`String`) or `Boolean`. Only one value is allowed per measurement (i.e. one record cannot store both `Integer` and `String` values for example. In addition a Measurement also needs to have a reference to a strategy.

A Measurement can be created as follows whereas the time stamp will be set according to the current system time:

```
getMeasurementService().createMeasurement(strategyName, "myMeasurement", 12.12345);
```

In addition a Measurement can also be created by providing an explicit time stamp:

```
ZoneDateTime zonedDateTime = ZonedDateTime.of(2019, 4, 1, 0, 0, 0, 0, ZoneId.of("UTC"));
getMeasurementService().createMeasurement(strategyName, "myMeasurement", zonedDateTime, 12.12345);
```

A Measurement can be deleted by using the following method:

```
getMeasurementService().deleteMeasurement(measurementId);
```

To read Measurements from the database the *Section 7.2.18, “Lookup Service”* has to be used which provides various Measurement lookup methods, e.g. `getMeasurementByMaxDate` or `getAllMeasurementsByMaxDate`.

---

**Note**

Only one measurement entry per strategy and name is allowed for a given time stamp. If there is an existing measurement with matching strategy, name and time stamp then creating a new measurement with new value will overwrite the previous value.

Only one value per
7.2.10. Option Service

AlgoTrader has full support for Option based trading strategies including an Option pricing engine. For further details see Chapter 15, Options & Futures.

7.2.11. Order Service

The OrderService is responsible for sending orders to brokers and exchanges in live trading as well as sending orders to the internal Section 5.1, “Exchange Simulator” during back tests. For further details please see Chapter 17, Order Management.

7.2.12. Portfolio Service

Financial valuations (strategy and position level) are available through the PortfolioService.

Since some values (e.g. market value) depend on whether the position is long or short, aggregated position values of different strategies for the same security cannot be retrieved just by adding position values from the corresponding strategies. Example:

- Security: VIX Dec 2012
- Current Bid: 16.50
- Current Ask: 16.60
- Strategy A: quantity +10 -> market value: 10 * 1000 * 16.50 = 165'000
- Strategy B: quantity -10 -> market value: 10 * 1000 * 16.60 = -166'000

The sum of above market values would be -1'000 which is obviously wrong.

As a consequence the PortfolioDAO provides lookup-methods that aggregate positions from the same security (of different strategies) in the correct manner (e.g. findOpenPositionsAggregated).

---

**Warning**

Positions are derived by taking all Transactions of the given Security and Strategy into account. It is therefore important not to modify Position entries directly in the database. In case transactions are added or modified manually to the database, please use the management action reset position and cash balances in the Figure 11.3, “AlgoTrader UI Settings”

- Portfolio Value logging to the database in Live-Trading Mode
- Portfolio Value logging to a .csv file in Simulation Mode
- Back test report and performance statistics display at the end of a simulation run (see Section 5.5, “Performance Statistics”)
• Portfolio Value Restoration Feature

All Performance Measurement features depend mainly on the Entity Portfolio Value which has the following attributes

• dateTime
• netLiqValue
• marketValue
• realizedPL
• unrealizedPL
• cashBalance
• openPositions
• leverage
• cashFlow (optional)

7.2.12.1. Portfolio Value Logging

In Live-Trading Mode Portfolio Values are recorded to the database for all running strategies on an hourly basis. In addition Portfolio Values are recorded every time a transaction occurs that influences performance. For the AlgoTrader Server these are Credits and Debits and for strategies, these are Rebalances. The corresponding value of the transaction is recorded in the optional attribute cashFlow of the Portfolio Value.

In Simulation Mode Portfolio Values are recorded to the file PortfolioReport.csv through the PortfolioReport class on a daily-basis using the AlgoTrader Reporting Functionality, see: Chapter 31, Reporting.

7.2.12.2. Portfolio Value Restoration Feature

In case a transaction that influences performance needs to be recorded for a prior period in time, all Portfolio Values since that time period are invalid and need to be restored.

Through the class RestorePortfolioValueStarter which uses PortfolioPersistenceService.restorePortfolioValues() Portfolio Values can be restored for the specified strategy and time period. For the restoration of each Portfolio Value all corresponding transactions up to that time have to be evaluated.

Note

The Portfolio Value restoration can take a considerable amount of time to complete.
7.2.13. Position Service

The PositionService provides the following position related methods:

- closePosition closes a single position
- closeAllPositions closes all positions in the system
- reducePosition reduces a position by the specified quantity
- transferPosition transfers a position from one strategy to another
- resetPositions calculates all Position based on Transactions in the database and makes adjustments if necessary.

Note

Closing and Reducing a position through the PositionService requires the definition of an order_preference with the name DEFAULT. For further details see Section 17.2.1, "Order Preferences".

The default order preference also includes an account, which means this feature is typically only usable with one account/adapter. If more than one account is in use, positions should be closes through the Section 7.2.11, "Order Service" by sending an order with a quantity that will offset the current position.

7.2.14. Property Service

The PropertyService can be used to assign arbitrary properties to the following classes:

- Account
- Exchange
- Order
- OrderPreference
- Position
- Security
- SecurityFamily
- Strategy
- Subscription
• Transaction

These classes are derived from the abstract class PropertyHolder. One or more Properties can be assigned to them. A Property can be any Java type (including Integer, Double, BigDecimal, String, Date, Boolean, etc.) but also arbitrary Java objects as long as the type implements Serializable.

Using the PropertyService a Property can be added as follows:

```java
getPropertyService().addProperty(StrategyImpl.class, 12, "myPropertyName", 12.12345);
```

The above example will add a Property named myPropertyName and value = 12.12345 to Strategy with id = 12.

Adding a custom object (implementing Serializable) is possible also:

```java
MyObject myObject = new MyObject("abc", 123, 22.44);
getPropertyService().addProperty(StrategyImpl.class, 12, "myPropertyName", myObject);
```

A Property can be deleted by using the following method:

```java
getPropertyService().removeProperty(StrategyImpl.class, 12, "myPropertyName");
```

Properties are available on the corresponding PropertyHolder objects as follows:

```java
double value = strategy.getProperty("myDoubleProperty", Double.class);
// or
double value = (Double)strategy.getProperty("myTextProperty");
```

All properties assigned to a particular PropertyHolder can be retrieved as follows:

```java
Map<String, Object> properties = strategy.getProperties();
```

7.2.15. Reference Data Service

Amongst others reference Data consists of static data like Security, SecurityFamily, SecurityReference, Account Entities. Reference Data can either be configured in the database directly through the corresponding tables, one can use the ReferenceDataService and corresponding ReferenceDataStarter, or conveniently the Reference Data Manager UI. For further details see Chapter 20, Reference Data

7.2.16. Rfq Service

AlgoTrader supports Request for Quote process. The RfqService provides the following methods:
• **sendRfq** sends a request for quote

• **getQuote** retrieves a quote if it hasn’t expired

• **discardQuotes** discard quotes.

---

**Note**

Accepting quote / placing order is possible with Order Service similarly like in Chapter 17, Order Management. Use `PreviouslyIndicatedOrderVOBuilder` and method `setQuoteId` to link the order to the received quote.

---

### 7.2.17. Market Data Cache Service

The `MarketDataCacheService` is intended to provide current market data and exchange rates to the strategy. the `MarketDataCacheService` keeps a local copy of each subscribed Security.

To access the last traded price of an instrument one can use the following code inside strategies:

```java
TickVO tick = (TickVO) getMarketDataCacheService().getCurrentMarketDataEvent(securityId);
BigDecimal lastPrice = tick.getLast();
```

To access the current exchange rate between USD and EUR one can use the following code inside strategies:

```java
double rate = getMarketDataCacheService().getForexRate(Currency.USD, Currency.EUR);
```

---

### 7.2.18. Lookup Service

the `LookupService` provides a large number of lookup methods for all objects available in the database. Examples:

• **getSecurityBySymbol** gets a security by its symbol

• **getExchangeByCode** gets an Exchange by its exchange code

• **getPositionBySecurityAndStrategy** gets a Position by Security and Strategy

• **getOpenPositionsByStrategy** gets open Positions for the specified Strategy

• **getAccountByName** gets an Account by its name

In addition to standard lookup methods above the `LookupService` also provides the following the generic lookup methods `find` and `findUnique` that can be used in situations where a standard lookup method is not available. These methods can be used as follows:
String query = "from StrategyImpl where name = :strategyName";
NamedParam param = new NamedParam("strategyName", "ABC");

Strategy strategy = getLookupService().find(Strategy.class, query, QueryType.HQL, false, param);

Please consult the JavaDoc\(^2\) for a full list of available methods.

In order to minimize the number of hits to the database the LookupService uses various levels of caching when reading from the database.

### 7.2.19. Strategy Service & Config Aware Strategy Service

All strategy main classes need to either extend StrategyService or ConfigAwareStrategyService. The ConfigAwareStrategyService provides the same functionality as the StrategyService but in addition also provides a reference to the strategy config object. For further details see Chapter 4, Strategy Development.

### 7.2.20. Subscription Service

The Subscription service allows a strategy to subscribe to Level 1/Level 2 market data and Aggregated order book. For that purpose the service provides several methods:

To subscribe for Level 1 market data use the following method:

```
getSubscriptionService().subscribeMarketDataEvent(strategyName, securityId, adapterType);
```

Note

The adapterType specifies the adapter to use when subscribing for market data (e.g. IB_NATIVE specifies the InteractiveBrokers native API adapter)

To subscribe for Level 2 market data use:

```
getSubscriptionService().subscribeToOrderBook(strategyName, securityId, adapterType);
```

To subscribe for Aggregated order book use:

```
getSubscriptionService().subscribeToAggregatedOrderBook(strategyName, symbol, securityClass);
```

\(^2\) [http://doc.algotrader.com/javadoc/ch/algotrader/service/LookupService.html](http://doc.algotrader.com/javadoc/ch/algotrader/service/LookupService.html)
Note

The `securityClass` specifies the type of security to subscribe, e.g. `Forex.class`.

Upon subscription, market data will be fed to the trading strategy that initiated the market data subscription. Market data will be fed to the corresponding Section 4.3.1.2, “Event Handler Methods” (e.g. `onBar`, `onTick`, `onOrderBook`) and also into the Esper Engine (if using Esper) where they are available as `Bar` and `Tick` events.

To unsubscribe Level 1 market data use the following method:

```
getSubscriptionService().unsubscribeMarketDataEvent(strategyName, securityId, adapterType);
```

To unsubscribe Level 2 market data use:

```
getSubscriptionService().unsubscribeFromOrderBook(strategyName, securityId, adapterType);
```

To unsubscribe an aggregated order book use:

```
getSubscriptionService().unsubscribeFromAggregatedOrderBook(strategyName, symbol, securityClass);
```

The `SubscriptionService` also supports the subscription for `GenericEvents`, see Section 18.6, “Generic Events”.

7.2.21. Reset Service

The `ResetService` can be used to reset the state of the database to a pre-defined state either before a simulation or if a reset of live trading is required.

To reset a live trading system multiple types of resets can be specified to the `reset` method using the `Enumeration` `ResetType`:

- **TRADES**
  - deletes all transactions (except the initial CREDIT)
  - resets all cash balances (except the one associated with the initial CREDIT)
  - deletes all positions

- **ORDERS**
  - delete all orders, order stats as well as order properties
SUBSCRIPTION
   deletes all subscriptions

COMBINATIONS_AND_COMPONENTS
   deletes combinations and components

MEASUREMENTS
   deletes measurements

PORTFOLIO_VALUES
   deletes portfolio values

OPTIONS
   deletes all options

FUTURES
   deletes all futures

MARKET_DATA
   deletes all bar and tick data

The method resetSimulation will reset the following items before each Simulation: Trades, Subscriptions, Combinations/Components, Properties, Options (if option prices are simulated) and Futures (if future prices are simulated).

7.3. Value Object

In contrast to Entities which are used to persist information, Value Objects are typically used for transmitting objects (e.g. via JMS or RMI). For each Entity a corresponding Value Object is generated. Value Objects are immutable (i.e. all fields are final and need to be set through the constructor)

Each Entity contains an inner Converter class that can be used to convert the Entity to its corresponding Value Object.

In addition to Value Objects ValueObjectBuilders exist which help creating Value Objects. Example:

```java
MarketOrderVO order = MarketOrderVOBuilder.create()
   .setStrategyId(strategyId)
   .setAccountId(accountId)
   .setSecurityId(securityId)
   .setQuantity(quantity)
   .setSide(side)
   .build();
```

For a full list of all Value Objects please visit our [JavaDoc](http://doc.algotrader.com/javadoc/ch/algotrader/vo/package-frame.html)

---

7.4. Enumerations

For selectable items with a fixed number of choices AlgoTrader contains Java 5 Enumerations. For a full list of all Enumerations please visit our JavaDoc\(^4\)

\(^4\) http://doc.algotrader.com/javadoc/ch/algotrader/enumeration/package-frame.html
Esper Engine

AlgoTrader uses the CEP (Complex Event Processing) engine Esper\(^1\). AlgoTrader based strategies can optionally make use of a dedicated Esper engine in addition to the Esper engine used by the AlgoTrader server.

8.1. Esper Introduction

Esper is an Event Stream Processing (ESP) and event correlation engine (CEP, Complex Event Processing). Targeted to real-time Event Driven Architectures (EDA), Esper is capable of triggering custom actions written as Plain Old Java Objects (POJO) when event conditions occur among event streams. It is designed for high-volume event correlation where millions of events coming in would make it impossible to store them all to later query them using classical database architecture.

A tailored Event Processing Language (EPL) allows expressing rich event conditions, correlation, possibly spanning time windows, thus minimizing the development effort required to set up a system that can react to complex situations.

Esper is a lightweight kernel written in Java which is fully embeddable into any Java process. It enables rapid development of applications that process large volumes of incoming messages or events.

8.1.1. Introduction to event streams and complex events using Esper

Information is critical to make wise decisions. This is true in real life but also in computing, and especially in the finance and trading area. Information flows in from different sources in the form of messages or events (e.g. market data events), giving a hint on the state at a given time such as stock price. That said, looking at those discrete events is most of the time meaningless. A trader needs to look at the stock trend over a period, possibly combined with other information to make the best deal at the right time.

While discrete events when looked one by one might be meaningless, event streams (i.e. an infinite set of events) considered over a sliding window and further correlated, are highly meaningful, and reacting to them with the minimal latency is critical for effective action and competitive advantage.

Relational databases or message-based systems such as JMS make it really hard to deal with temporal data and real-time queries. Indeed, databases require explicit querying to return meaningful data and are not suited to push data as it changes. JMS systems are stateless and require the developer to implement the temporal and aggregation logic himself. By contrast, the Esper engine provides a higher abstraction and intelligence and can be thought of as a database turned upside-down: instead of storing the data and running queries against stored data, Esper allows applications to store queries and run the data through. Response from the Esper engine is real-time when conditions occur that match user defined queries. The execution model is thus continuous rather than only when a query is submitted.

In Esper, a tailored EPL allows registering queries in the engine. A listener class, which is basically a POJO, will then be called by the engine when the EPL condition is matched as events flow in. The EPL enables to

---

\(^1\) [http://www.espertech.com/esper/]

\(^2\) Most of this section has been reproduced from the Esper website
express complex matching conditions that include temporal windows, joining of different event streams, as well as filtering, aggregation, and sorting. Esper statements can also be combined together with “followed by” conditions thus deriving complex events from more simple events. Events can be represented as JavaBean classes, legacy Java classes, XML document or java.util.Map, which promotes reuse of existing systems acting as messages publishers.

A trivial yet meaningful example is as follow: assume a trader wants to buy Google stock as soon as the price goes below some floor value, not when looking at each tick but when the computation is done over a sliding time window, say of 30 seconds. Given a TickVO event bean with a last price field and a reference to a Security ID and the following EPL, a listener POJO would get notified as ticks come in to trigger the buy order:

```java
select
    avg(last)
from
    TickVO.win:time(30 sec)
where
    securityId=12
```

### 8.1.2. Event representations

Java classes are a simple, rich and versatile way to represent events in Esper. Java classes offer inheritance and polymorphism via interfaces and super-classes, and can represent a complex business domain via an object graph. In AlgoTrader event class like TickVO, BarVO, OrderVO, OrderStatusVO etc. are made available to Esper engines by default. In addition any arbitrary java class can be used inside Esper engines after declaring them.

In addition to Java classes, Maps and XML are an alternative way of representing events.

### 8.1.3. Event Stream Analysis

EPL statements derive and aggregate information from one or more streams of events, to join or merge event streams, and to feed results from one event stream to subsequent statements.

EPL is similar to SQL in it's use of the `select` clause and the `where` clause. However EPL statements instead of tables use event streams and a concept called views. Similar to tables in an SQL statement, views define the data available for querying and filtering. Views can represent windows over a stream of events. Views can also sort events, derive statistics from event properties, group events or handle unique event property values.

This is a sample EPL statement that computes the average of the last price for the last 30 seconds of Tick events:

```java
select
    avg(last)
from
    TickVO.win:time(30 sec)
```
A sample EPL that returns the average of the last price per symbol for the last 100 Ticks.

```epl
select
    securityId
    avg(last) as averagePrice
from
    TickVO.win:length(100)
group by
    securityId
```

This example joins 2 event streams. The first event stream consists of Bar events for which we keep the last 30 minutes (1800 seconds). The second stream is Tick events for which we consider the last 30 seconds. The streams are joined on `securityId`.

```epl
select
    bar.securityId as securityId,
    max(bar.high) as maxHigh,
    min(bar.low) as minLow,
    last(tick.last) as lastPrice
from
    BarVO.win:time(30 min) as bar,
    TickVO.win:time(30 sec) as tick
where
    bar.securityId = tick.securityId
```

### 8.1.4. Combining Pattern Matching with Event Stream Analysis

Patterns match when a sequence (or absence) of events is detected. Pattern match results are available for further analysis and processing.

The pattern below detects a situation where an `OrderStatus` event is not followed by another `OrderStatus` event corresponding to the same internal order id within 10 seconds. The statement further counts all such occurrences grouped per internal order id.

```epl
select
    a.intId,
    count(*)
from
    pattern [every a=OrderStatus
             -> (timer:interval(10 sec) and not OrderStatus(intId=a.intId))]
group by
    id
```
8.1.5. Named windows

A named window is a global data window that can take part in many statement queries, and that can be selected-from, inserted-into and deleted-from by multiple statements. Named windows are similar to a table in a relational database system.

One can create a named window for example as follows:

```plaintext
create window
  SecurityWindow
as
  (symbol String, triggerPrice double)
```

One can trigger a select, update or delete when an event arrives. Here we show a select that simply counts the number of rows:

```plaintext
on
  TriggerEvent
select
  count(*)
from
  SecurityWindow
```

Named windows can also be queried with fire-and-forget queries through
`ch.algotrader.esper.Engine.executeQuery` and
`ch.algotrader.esper.Engine.executeSingelObjectQuery`.

8.1.6. Variables

A variable is a scalar, object or event value that is available for use in all statements including patterns. Variables can be used in an expression anywhere in EPL.

8.2. Esper Quick Start Guide

This quick start guide provides step-by-step instructions for using Esper inside AlgoTrader.

8.2.1. Event Types

Java classes are a good choice for representing events, however Map-based or XML event representations can also be good choices depending on the architectural requirements.

AlgoTrader provides a number of Value Objects that can be used as Esper Events (e.g. `TickVO`, `BarVO`, `OrderStatusVO`, etc.)

3Most of this section has been reproduced from the Esper website.
8.2.2. Creating a Statement

A statement is a continuous query registered with an Esper engine instance that provides results to listeners as new data arrives, in real-time, or on demand via the iterator API (see `ch.algotrader.esper.Engine.executeQuery`).

The next code snippet shows an Esper module containing a continuous query. The query returns the average price over all `TickVO` events that arrived in the last 30 seconds:

```sql
select
  avg(price) as avgPrice
from
  TickVO.win:time(30 sec)
```

Each of the Esper engines inside AlgoTrader can contain several modules. Modules specified through the `initModules` and `runModules` attribute of the Esper Engine Spring Bean Definition are loaded automatically on start-up.

```xml
<bean id="testEngine" class="ch.algotrader.esper.EngineFactoryBean">
  <property name="strategyName" value="TEST"/>
  <property name="configResource" value="esper-test.cfg.xml"/>
  <property name="configParams" ref="testConfigParams"/>
  <property name="initModules" value="market-data"/>
  <property name="runModules" value="order-handling"/>
</bean>
```

Note

A module definition of `market-data` will look for a module file called `module-market-data.epl`.

Additional modules can be deployed at runtime using the method `Engine.deployModule`.

8.2.3. Adding a Subscriber

A subscriber object is a direct binding of query results to a Java object. The object, a POJO, receives statement results via method invocation. The subscriber class does not need to implement an interface or extend a superclass. Only one subscriber object may be set for a statement.

Subscriber objects have several advantages over listeners. First, they offer a substantial performance benefit: Query results are delivered directly to the Java method(s) through Java virtual machine method calls, and there is no intermediate representation (`EventBean`). Second, as subscribers receive strongly-typed parameters, the subscriber code tends to be simpler.
The subscriber class must provide a public method to receive events. The number and types of parameters
declared by the update method must match the number and types of columns as specified in the select clause,
in the same order as in the select clause.

For the following statement:

```java
@Subscriber(className='ch.algotrader.listener.MySubscriber#process')
select
    orderId, price, count(*)
from
    OrderEvent;
```

the Subscriber class looks as follows:

```java
public class MySubscriber {
    public void process(String orderId, double price, long count) {
        System.out.println("orderId=" + orderId + ",price=" + price + ",count=" + count);
    }
}
```

8.2.4. Adding a Listener

Listeners are invoked by the engine in response to one or more events that change a statement's result set.
Listeners implement the `UpdateListener` interface and act on `EventBean` instances as the next code snippet outlines:

```java
public class MyListener implements UpdateListener {
    public void update(EventBean[] newEvents, EventBean[] oldEvents) {
        System.out.println("avg=" + newEvents[0].get("avgPrice"));
    }
}
```

By attaching the listener to the statement via the following annotation the engine provides the statement's results to the listener:

```java
@Listeners(classNames={'ch.algotrader.listener.MyListener'})
select
    avg(price) as avgPrice
from
    TickVO.win:time(30 sec)
```
8.2.5. Sending events

The runtime API accepts events for processing. As a statement’s results change, the engine indicates the new results to listeners right when the events are processed by the engine.

Incoming market data events (e.g. Ticks), submitted Orders, OrderStatus events, received Fills, etc are automatically sent into the corresponding Esper engines.

Additionally custom events can be sent into an Esper engine. The following code snipped creates an arbitrary event and sends it into the Esper engine instead an AlgoTrader strategy named EXAMPLE.

```java
MyEvent event = new MyEvent("TEST_EVENT");
engine.sendEvent(event);
```

8.2.6. Configuration

Esper Configuration helps make statements more readable and provides the opportunity to plug-in extensions.

Each Esper Engine loads the default esper-common.cfg.xml file. In addition strategies load all Esper configuration files named esper-xxx.cfg.xml in the class path. This configuration file defines settings like:

- Event Types
- Auto Import Classes & Packages
- Custom Aggregation Functions
- Variables
- General Engine Settings

8.3. Esper Documentation

Esper provides in depth documentation.

The following chapters of the Esper Documentation are relevant for developing trading strategies with AlgoTrader based on Esper:

- 2. Event Representations
- 3. Processing Model
- 5. EPL Reference: Clauses

---

8.4. AlgoTrader specific Esper Artifacts

8.4.1. Engine & EngineManager

Inside AlgoTrader Esper engine instances are wrapped by the `EngineImpl` which implements the `Engine` interface. Individual Engine instances can be located through the `EngineManager` singleton.

8.4.1.1. Engine

The `Engine` interface has methods available for the following tasks:

- deployment / un-deployment of Esper statements and modules
- sending events
- execute fire-and-forget queries
- retrieve current statement state
- management of the Esper clock
- synchronized processing (coordination) of events from different sources into the Esper engine
- management of Esper variables
- adding Section 8.4.8, “Callbacks”

For further information please visit the relevant JavaDoc.

For testing purposes there is an abstract do-nothing implementation of the `Engine` interface available named `AbstractEngine`.

8.4.1.2. EngineManager

The `EngineManager` interface represents the main entry point to different Engines running inside the JVM. The `EngineManager` has methods available for the following tasks:

---

• Lookup of available Engines
• Query of the current Engine time
• Management of statement metrics

For further information please visit the relevant JavaDoc.

---

**Note**

Engine instances are managed and configured through Spring configuration. Engines are standard Spring managed beans that get automatically registered with EngineManager upon startup.

---

### 8.4.2. Modules

#### 8.4.2.1. AlgoTrader Server Modules

The AlgoTrader Server contains the following Esper modules:

#### Table 8.1. AlgoTrader Server modules

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module-algo-xxx.epl</td>
<td>Each Execution Algo has its own module</td>
</tr>
<tr>
<td>module-combination.epl</td>
<td>Combination / Component related functionality (see Chapter 25, Synthetic Securities and Derivative Spreads)</td>
</tr>
<tr>
<td>module-current-values.epl</td>
<td>Store current market data values</td>
</tr>
<tr>
<td>module-ib.epl</td>
<td>IB specific statements</td>
</tr>
<tr>
<td>module-market-data.epl</td>
<td>Statements related to market data</td>
</tr>
<tr>
<td>module-metrics.epl</td>
<td>Statements needed for Engine Metrics</td>
</tr>
<tr>
<td>module-performance.epl</td>
<td>Evaluation of performance metrics</td>
</tr>
<tr>
<td>module-portfolio.epl</td>
<td>Portfolio management functions</td>
</tr>
<tr>
<td>module-prepared.epl</td>
<td>Prepared Statements available to strategies</td>
</tr>
<tr>
<td>module-server-prepared.epl</td>
<td>Server-Side prepared Statements</td>
</tr>
<tr>
<td>module-trades.epl</td>
<td>Statements related to orders and executions</td>
</tr>
</tbody>
</table>

---

**Note**

init and run modules of the AlgoTrader Server can be defined through config properties in conf-core.properties.
8.4.2.2. Strategy Modules

Strategies are completely free in the definition of their Esper Statements. Examples of Statements used by strategies are:

- Creation of technical indicators (e.g. Moving Average, Bollinger Bands, ATR, etc.)
- Creation of trade signals
- Trend evaluation
- Open / Close / Increase / Reduce Positions
- Roll Positions (for Options and Futures)
- Pattern recognition

**Note**

It is generally recommended to use Esper statements for anything up to signal generation but then use Java for execution of actions (e.g. send an order, set a stop or close a position).

8.4.3. Tags

In addition to the *Esper standard tags*\(^\text{13}\) the following tags are available:

**Table 8.2. Esper tags**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Condition(key='xxx')</td>
<td>Statement is only deployed if defined configuration parameter is set to &quot;true&quot;</td>
</tr>
<tr>
<td>@SimulationOnly</td>
<td>Statement is only deployed in simulation</td>
</tr>
<tr>
<td>@RunTimeOnly()</td>
<td>Statement is only deployed in Live-Trading mode</td>
</tr>
<tr>
<td>@Listeners(classNames={'...'})</td>
<td>attaches one or multiple listeners to the statement</td>
</tr>
<tr>
<td>@Subscriber(className='...')</td>
<td>attaches a subscriber to the statement</td>
</tr>
</tbody>
</table>

8.4.4. Subscribers

The system provides the following Subscribers out-of-the-box:

- **IndicatorSubscriber**
  
  Prints all values as a comma-separated-list (CSV) to the file *files/report/IndicatorReport.csv* (Headers are not available).

Listeners

TestSubscriber
Prints all values to the Log by using the `toString` method of the event object.

VoidSubscriber
Do-nothing subscriber, useful when select clauses call static methods

ExceptionSubscriber
Prints a value as an Error to the Log.

Any public method of a component defined in the Spring application context can potentially be used as a subscriber provided the parameter signature of the method can be supported by Esper: One can define a subscriber by specifying a Spring bean name followed by hash (#) followed by a method name exposed by this bean.

```java
@Name('TAKE_PROFIT')
@Subscriber(className='boxService#takeProfit')
```

The syntax also supports property placeholder expansion. This can especially useful when using multiple instances of the same strategy in the same JVM process.

```java
@Name('TAKE_PROFIT')
@Subscriber(className='${strategyName}Service#takeProfit')
```

8.4.5. Listeners

The system provides the following Listeners out-of-the-box:

IndicatorListener
Prints all values as a comma-separated-list (CSV) to the file `files/report/IndicatorReport.csv`. Headers will be extracted from the supplied Statement.

RendererListener:
Prints all values to the Log in XML format.

TestListener
Prints all values to the Log by using the `toString` method of the event object.

StatementAwareTestListener
Prints all values including the statement name to the Log by using the `toString` method of the event object.

8.4.6. Service method invocation in Esper scripts

AlgoTrader Esper statements can access standard platform services directly

By default AlgoTrader exposes the following services to Esper statements, which strategies can make use of:
• LookupService
• PortfolioService
• CalendarService
• OrderService
• PositionService
• MarketDataService
• OptionService

Here are some examples. To retrieve a security based on its `securityId` one can use the following statement:

```java
select bar, security
from BarVO as bar unidirectional,
method:lookupService.getSecurity(bar.securityId) as security;
```

To retrieve an open order by its `intId` one can use the following statement:

```java
select fill, openOrder
from FillVO as fill unidirectional,
method:orderService.getOrderByIntId(fill.orderIntId) as openOrder;
```

### 8.4.7. Aggregation Functions

The system provides the following custom aggregation functions:

#### 8.4.7.1. ExponentialMovingAverage

The `ExponentialMovingAverageFunction` can be used to construct an exponential moving average of a time series, e.g.:

```java
select ema(last, 10) from TickVO;
```
This will create a 10-period exponential moving average of the Tick last price.

### 8.4.7.2. GenericTALibFunction

The `GenericTALibFunction` is a port of `ta-lib`\(^{14}\) to AlgoTrader. It supports all TA-Lib operations.

Please consult [TA-Lib\(^{15}\)](http://doc.algotrader.com/ta-lib.html) for a list of all TA-Lib methods and their parameters.

If the TA-Lib Function returns just one value, the value is directly exposed by the `AggregationFunction`.

Example: The TA-Lib function `movingAverage` has just one double typed return value which can be accessed directly.

```sql
insert into MovingAverage
select talib("movingAverage", close.doubleValue(), 30, "Sma") as result
from BarVO;

select result
from MovingAverage;
```

If the TA-Lib Function returns multiple-values, a dynamic class will be generated on the fly, which gives access to properly typed return-values. All return value names are lower-case!

Example: The TA-Lib function `stochF` has return values: `outFastK` and `outFastD`. The returned dynamic class will have double typed properties by the name of: `fastk` and `fastd` (all lowercase).

```sql
insert into Stochastic
select talib("stochF", high.doubleValue(), low.doubleValue(), close.doubleValue(), 3, 2, "Sma") as result
from BarVO;

select result.fastk, result.fastd
from Stochastic;
```

Some functions are influenced by the entire range of the past data. These functions are sometimes called functions with memory. An example is the EMA (Exponential Moving Average). For these functions an optional unstable period parameter can be specified. The following statement will create a 30 period moving average with an unstable period of 10.

```sql
insert into MovingAverage
select talib("movingAverage", close.doubleValue(), 30, "Ema", 10) as result
from BarVO;
```

\(^{14}\) [http://ta-lib.org/]

\(^{15}\) [http://doc.algotrader.com/ta-lib.html]
For further details about the unstable period please see: SetUnstablePeriod\textsuperscript{16}

For additional information please visit the corresponding JavaDoc\textsuperscript{17}.

\section*{Note}
As an alternative to the ta-lib based exponential moving average function the Esper aggregation function \textsection{8.4.7.1, “ExponentialMovingAverage”} can be used which keeps the entire history and not just the unstable period.

\section*{8.4.8. Callbacks}

\subsection*{8.4.8.1. First tick callback}

A \texttt{BiConsumer<String, List<TickVO>>> function can be registered with the Esper engine using the Engine\#addFirstTickCallback() method. Whenever at least one Tick of each specified security has arrived the consumer will be executed receiving the name of the strategy and a list of ticks as input.

A typical use case of a first tick callback looks like this:

\begin{verbatim}
engine.addFirstTickCallback(securityIds, (strategyName, ticks) -> {
    ...
});
for (long securityId : securityIds) {
    getSubscriptionService().subscribeMarketDataEvent(strategyName, securityId);
}
\end{verbatim}

\subsection*{8.4.8.2. Trade callback}

A \texttt{BiConsumer<String, List<OrderStatusVO>>> function can be registered with the Esper engine using the Engine\#addTradeCallback() method. Whenever all corresponding orders have been fully executed or canceled the consumer will be executed receiving the name of the strategy and a list of order status messages as input.

In order to correctly associate the trade callback with a specific order an orderId has to be retrieved from the order service and set onto the order before attaching the trade callback.

A typical use case of a trade callback looks like this:

\begin{verbatim}
String orderId = getOrderService().getNextOrderId(order.getClass(), accountId);
\end{verbatim}

\footnotesize
\textsuperscript{16} https://ta-lib.org/d_api/ta_setunstableperiod.html
\textsuperscript{17} http://doc.algotrader.com/javadoc/ch/algotrader/esper/aggregation/GenericTALibFunction.html
order.setIntId(orderId);

engine.addTradeCallback(Collections.singleton(orderId), (strategyName, orderStatus) -> {
  ...
});

getOrderService().sendOrders(orders);

The `Engine#addFullExecutionCallback()` method can be used to register a callback logging an error if the order did not execute fully.

String orderId = getOrderService().getNextOrderId(order.getClass(), accountId);

order.setIntId(orderId);

engine.addFullExecutionCallback(Collections.singleton(orderId));

getOrderService().sendOrders(orders);

Note

With Fix based Broker / Exchange connections the `TradeCallback` only works with the initial order but not with any subsequent order modifications.

8.4.8.3. Trade persisted callback

A `Consumer<List<OrderCompletionVO>>` function can be registered with the Esper engine using the `Engine#addTradePersistedCallback()` method. Whenever all corresponding orders have been fully executed and all corresponding database transactions (e.g. `OrderStatus`, `Transaction` and `Position`) have been fully executed the consumer will be executed receiving a list of `OrderCompletionVO` objects as input.

This callback is particularly useful for situations where one needs to have a guarantee that all order related database transactions have been fully executed before continuing with next steps, e.g. to retrieve the current position quantity after an order has been executed. If using a regular trade callback for this, the Position Entity might not have been fully persisted by the time the consumer is executed. However when using the trade persisted callback it is guaranteed that the Position Entity has been fully updated in the database.

In order to correctly associate the trade callback with a specific order an `orderId` has to be retrieved from the order service and set onto the order before attaching the trade callback.

A typical use case of a trade persisted callback looks like this:

String orderId = getOrderService().getNextOrderId(order.getClass(), accountId);
order.setIntId(orderId);

engine.addTradePersistedCallback(Collections.singleton(orderId), orderCompletions -> {
    // ...
});

getOrderService().sendOrders(orders);

---

**Note**

The Trade persisted callback is only supported in runtime mode but it is not supported in simulation mode. It is recommended to use the Section 8.4.8.2, "Trade callback" instead in simulation mode. The Trade persisted callback will only get executed if there has been at least one (partial) execution but not if an order has been cancelled or rejected before there has been any execution.

---

### 8.4.8.4. Open / close position callback

A `Consumer<PositionMutationVO>` function can be registered with the Esper engine using the `Engine#addOpenPositionCallback()` or `Engine#addClosePositionCallback()` methods. Whenever a corresponding transaction causes a new position to open / close the consumer will be executed receiving `PositionMutationVO` as input.

```
engine.addOpenPositionCallback(securityId, positionMutation -> {
    // ...
});

engine.addClosePositionCallback(securityId, positionMutation -> {
    // ...
});
```

---

**Note**

It is guaranteed that the position is fully persistent to the database by the time the consumer is called.

---

### 8.4.8.5. Timer callback

A `Consumer<Date>` function can be registered with the Esper engine using the `Engine#addTimerCallback()` method. Whenever the system time has reached the specified time the consumer will be executed receiving the actual time as input. To distinguish multiple timers from each other an additional `name` parameter is available to name each timer instance.
8.5. Esper Threading

Esper has several options for enabling a multi-threaded environment, see Esper API Threading\(^\text{18}\).

In Live-Trading Mode AlgoTrader uses outbound threading with 3 threads by default. This means that all Subscriber / Listener Tasks are handled by a thread-pool of 3 threads. The number of outbound threads can be changed inside conf.properties or via Section 2.4, “VM Options”:

```
# number of Esper outbound threads to be used in Live Trading Mode for the Server Engine
misc.outboundServerEngineThreads = 3

# number of Esper outbound threads to be used in Live Trading Mode for the Strategy Engines
misc.outboundStrategyEngineThreads = 3
```

For debugging reasons AlgoTrader logs the name of the thread using log4j, see Chapter 30, Logging.

Caching

AlgoTrader uses an in-memory data-grid solution called Hazelcast\(^1\). This layer resides between the application layer (business logic) and persistence. It is responsible for providing easy and fast access to any persistent data for both read and write operations.

9.1. Hazelcast Introduction

Note

The Hazelcast product family offers various features and software packages. Algotrader relies on the IMDG product.

As a general rule we can assume, that there is at least *1 magnitude difference*\(^2\) in speed between reading from RAM and from disks. Algotrader uses in-memory caching not only to provide clear separation between persistence and application layer, but to be able to perform high-speed trading without being bound to disk speed. This approach however requires some changes to what could be considered "standard" way of implementing persistence in Java applications.

9.1.1. Persistence

The caching layer separates the application layer from the persistence logic. The application layer does not directly access DAO methods anymore and is required to go through CacheFacade classes to access data. These CacheFacade classes internally rely on Hazelcast IMaps\(^3\). IMaps have similar characteristics to the standard Java Map interface. The biggest difference being the possibility to attach MapStores to these maps, giving them the ability to use persistent storage.

9.1.1.1. Object state

It is important to remember that objects in the Algotrader application layer are no longer attached to any Hibernate session. They get detached on read and merged back to the persistence context upon save. This also means that any mutation on these object have to be explicitly saved or otherwise the change will not be persisted. Objects served out of caches are - by default - not shared and will be created by deserialization whenever requested.

9.1.1.2. ID assignment

In a traditional environment objects belonging to the relational database often get an ID assigned to them by the database itself via auto-generated fields or sequences. In Algotrader's case this is no longer true, since we assign these IDs prior to put them into caches. For this purpose we use the `AutoIncrementIdGenerator`

\(^1\) https://hazelcast.org/imdg/features/

\(^2\) https://colin-scott.github.io/personal_website/research/interactive_latency.html

\(^3\) https://docs.hazelcast.org/docs/3.8/javadoc/com/hazelcast/core/IMap.html
class. This solution allows us to generated IDs prior to saving objects. Please note that this ID pool is shared. This means that after having persisted a set of objects in one table (order for example), the next ID assigned for another table (transaction) will be consecutive from the one last seen on the previous table. It is not advised to assign IDs yourself.

9.1.1.3. Asynchronous persistence

In order to drastically increase the throughput of the system (both live and simulation), Algotrader halts the application logic only the smallest amount of time possible to save its current state. Most trading applications would suffer the largest part of their latency due to having to save their internal state (be that, orders, positions, transactions) to a storage before they can continue on further operations. Using Hazelcast caches Algotrader only stops for the time needed to save changes in memory, but will delegate the task of long-term storage to other threads, thus unblocking the main application threads. This async write behavior (often referred to as write-behind) is configurable per cache/table in Algotrader. For details see the configuration section.

9.1.1.4. Database constraints

Due to the async nature of the persistence layer, database constraints cannot be utilized the same way as for synchronous DB operations. The simplest example to highlight the problem is the relation between orders and transactions. A freshly created order might results in a new transaction after having received a fill. Saving this transaction however can be problematic now since there is no guarantee that the original order has been saved already since both of them being saved write-behind. To overcome this problem Algotrader does not rely on database level constraints. To provide the same level of correctness checking, we have a constraint checking system in place. This is enabled by default, but can be switched off in case in a certain situation (like backtesting) the potential corruption of the database state is not considered to be a real risk.

9.1.1.5. Persistence-less operation

In certain situations (backtesting/unit testing) truly persistent DB operations might not be needed as the data would be discarded anyway. In this case it is possible to omit the pooledDataSource profile from any Algotrader starter configuration. This will result in Algotrader starting up without any backing store, where data will only be kept in memory. Be careful: the application will have lose its working memory on restarts and there is no way to access that storage later.

9.1.2. Cache access

9.1.2.1. Diagnostics via JMX/Rest

Hazelcast my default makes many of its internal metrics available via JMX. These can be accessed by standard tools, like JVisualVM (shipped with most Java distributions) or libraries like Jolokia. They allow to query the internal state of caches and get relevant insights regarding current load, frequent and/or slow cache operations, etc.

Algotrader also exposes a set of cache operations that can be useful for debugging/operational purposes. These services are available locally under the following URL: http://localhost:9090/rest/cache/ on the machine Algotrader is running. Using these endpoints users can query in JSON format (get/getAll), toString format (print/
printAll) any caches that currently exists. There are also a number of non-readonly operations available, such as evict/delete/refresh etc. These perform the corresponding operation on the cache itself, and should be used with care. They are intended to be used during development phase, and primarily not in production.

9.1.2.2. Eviction and preloading

Certain caches can grow indefinitely in size and keeping all items in-memory is not practical. For this case Algotrader has a configurable eviction logic that comes with defaults but can be fine-tuned for individual cases. Users might choose to use this facility in case they have a large number of orders / transactions created for example.

During startup it is equally good idea to limit the number of entries that get preloaded into memory. It is not an issue for small caches (like strategy, account, etc) but makes sense not to load very old transaction or orders back into memory. These characteristics of each cache can be configured independently (see next section).

9.1.2.3. In-memory format

By default the caches store their content in a serialized format, using the highly efficient Kryo serialization package. This reduced their memory footprint drastically, however it has a significant drawback. In case the caches are often queried, this configuration requires the JVM to frequently deserialize its content. So in certain cases it makes more sense to not make use of the serialization at all and store objects in their original format. For small (and mostly static caches) this is the most efficient setup.

9.1.3. Configuration

9.1.3.1. Defaults

Algotrader comes with a set of generic cache-related configs and some others that are specific to individual caches.

Table 9.1. Cache configuration

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Unit</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hz.validateEntities</td>
<td>Whether the cache-level consistency check system is enabled or not</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>hz.mapDefaults.useInternalCache</td>
<td>To control whether identical elements can be served (true) from the internal cache or copies are needed</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>hz.mapDefaults.writeDelaySeconds</td>
<td>Maximum time before persistent write happens (0 means sync writing)</td>
<td>Integer</td>
<td>1</td>
</tr>
<tr>
<td>hz.mapDefaults.inMemoryFormat</td>
<td>Whether cache content is serialized by default or not</td>
<td>BINARY/OBJECT</td>
<td>BINARY</td>
</tr>
<tr>
<td>hz.mapDefaults.timeToLiveSeconds</td>
<td>Maximum time for a cached entry to stay in the cache (0 means infinite)</td>
<td>Integer</td>
<td>0</td>
</tr>
<tr>
<td>hz.mapDefaults.maxIdleSeconds</td>
<td>Maximum time for a cached entry to stay idle in the cache (0 means infinite)</td>
<td>Integer</td>
<td>0</td>
</tr>
</tbody>
</table>
9.1.3.2. Map-level configuration overrides

Algotrader comes with a set of generic cache-related configs (see above) and some others that are specific to individual caches. These can be individually overridden in case some special condition would require that. In that case individual caches names can be substituted into the default param names (see above), in the following format: `hz.maps.POSITION_MAP.writeDelaySeconds=0`. This line for example would make position changes persisted immediately in a synchronous manner.

**Warning**

Modifying these settings without a thorough understanding of their implications is dangerous. The values currently set are suitable for most cases, and most users should not need to ever change them. In case you are considering to change them, please consult the documentation first and if in doubt, consider raising a support ticket beforehand.
Database

10.1. Instances

AlgoTrader uses MySQL\(^1\) to store transaction data.

10.2. Flyway

AlgoTrader uses the database migration library flyway\(^2\) to keep databases in-line with the current version of AlgoTrader. Flyway executes database migration scripts to ensure that the database is in the state corresponding to the version of AlgoTrader installed.

Flyway is embedded into the AlgoTrader platform, and can perform a migration of the Database on each startup automatically. In case the configured schema does not exist on MySQL, it will create it. This schemas default-name is algotrader. The feature is activated on all startup-scripts including docker but deactivated in the development environment. misc.flywayMigrateOnStartup is the property to control this behaviour, it can be passed as a VM Argument.

```
-Dmisc.flywayMigrateOnStartup=true
```

In case you need to reset the Database, the easiest way is to delete the corresponding schema on MySQL and start AlgoTrader. For further migration related issues, a class within AlgoTrader ch.algotrader.starter.FlywayRunner can be used to execute other flyway commands. It is not recommended to use Flyways command-line-tool instead, because of potential problems when using several flyway versions on the same schema.

For additional information on flyway please visit the flyway documentation\(^3\).

The database migration scripts are located in /bootstrap/conf/flyway.

10.3. Files

The directory /bootstrap/conf contains all relevant database files:

- flyway/sql contains the flyway migration scripts that will be executed to initialize and update the MySQL database

- src/main/resources/db-samples/mysql database sample data for accounts, exchanges, securities (FX majors, S&P 500 & EURO STOXX 50 stocks), security families and order preferences.

---

\(^1\) [https://www.mysql.com/](https://www.mysql.com/)
\(^2\) [https://flywaydb.org/](https://flywaydb.org/)
\(^3\) [https://flywaydb.org/documentation/maven/](https://flywaydb.org/documentation/maven/)
10.4. Data Source

AlgoTrader provides different Data Source for different scenarios. One of the available Data Sources needs to be configured via the following VM argument:

```
-Dspring.profiles.active=<{dataSource}>
```

`dataSource` can be one of the following:

- **pooledDataSource**: A Data Source that uses connection pooling based on [C3P0](https://www.mchange.com/projects/c3p0/)
- **singleDataSource**: A Data Source that uses one single database connection and no connection pooling.
- **hybridDataSource**: A Data Source prepared for backtesting purposes, it keeps only reference data in physical DB (Components, Strategies, Exchanges, TradingHours, Holidays, Securities, SecurityFamilies, Accounts, OrderPreferences, SecurityReferences) Other, transactional data (Transactions, CashBalances, Positions, Orders, OrderStatuses, Subscriptions, PortfolioValues, Measurements, Quotes, QuoteRequests) are kept only in the cache and are erased after restart.
Client

AlgoTrader provides different types of clients all of which are targeting a different audience and use case:

- The html5 UI: used to monitor/manage trading activities while AlgoTrader is running
- IntelliJ IDEA: Contains the Strategy Creation Wizard, the Config Editor as well as the strategy simulation environment
- Reference Data Manager (RDM)
- Historical Data Manager (HDM)

11.1. HTML5 UI

Note

Officially AlgoTrader only supports Google Chrome. However, the AlgoTrader UI will most likely work with the most recent version of any modern browser.

The AlgoTrader UI provides the following features.

- Live Market Data updated in real-time.
- Tables showing current Orders, Transactions, Positions and Market Data.
- All tables provide (multicolumn) sorting, filtering, column selection and reordering, and scrolling.
- Display of Alarms and Notifications in case something unexpected happens.
- Supports multiple currencies and automatic currency conversion.
- Since the UI is based on HTML5, it can easily be integrated into corporate IT infrastructures using firewalls, VPNs, and remote locations.
- Auto-completion feature for security selection.

When AlgoTrader server starts, it automatically opens the client.

To manually open the client, point the browser to one of the following URL.

http://localhost:9090
11.1.1. Header and toolbar

Figure 11.1. AlgoTrader UI Header

Using the strategy selection menu located at the top of the screen (to the right), one can select either an aggregated view over the entire system or a strategy specific view.

When a single strategy is selected the client will show orders, transactions, positions and market data subscriptions related to the selected strategy only. When ALL is selected the client will show orders, transactions, positions and market data subscriptions for all strategies.
The header tiles show general figures (like Net Liquidity, Unrealized P&L, Cash, etc.). The default valuation currency is USD. You can change it by updating the `misc.portfolioBaseCurrency` value in the `conf.properties` file.

If one opens a settings menu form on bottom left corner one can see the following settings are available there:

1. Tiles - one can configure the visibility of general figures in header
2. Order defaults - default order related values like default quantity and default time-in-force
3. Data throttling - Sets the update interval of all tables, e.g. if the interval is 333ms, the tables will buffer all data updates and only make display changes every 333ms.
4. Order removal delay - Remove completed orders after x seconds
5. Source of chart's data - Preferably the source of chart's data is AT's own historical data source, if that's not present then TradingView's delayed data can be used.
6. Chart's timezone - Either exchange's timezone or AT's server timezone
7. Drawing on chart - Enable a menu on chart with drawing tools.
8. UI Theme - Change the theme to Dark mode or Blue or Light
9. Reset Position and Cash Balance - Resets all Cash Balances and Position quantities based on Transactions in the database and makes adjustments if necessary.
10. Restore default settings - clicking it will remove any custom UI settings saved in browser's storage. This includes visible columns, columns order and all settings described above.
**Figure 11.3. AlgoTrader UI Settings**

Notifications are displayed in the lower right hand side of the screen

---

**Tiles**
- Realized P&L
- Unrealized P&L
- Cash Balance
- Net Liquidity
- Positions
- Market Value

**Order Defaults**
- Quantity
- TIF
- Account Type: Margin

**UI**
- Data throttling: 333
- Order removal delay
- Source of chart's data: Trading View
- Chart's timezone: Exchange
- Drawing on chart: Enabled
- UI Theme: Light

**Restore & Reset**
- Position & balance: Reset
- Default UI settings: Restore

**Version**
- **UI**: 2020-06-10 22:36:52 +0200 - 83ff4343
- **BACKEND**: 

---

136
Figure 11.4. AlgoTrader UI Notification

There are three types of notifications: information-s (green), warnings (orange) and alerts (red). In addition to warnings and alerts appearing in bottom right corner, a bell icon will appear at the top left of the screen.

To open the list of all warnings and alerts click on the bell icon.

Figure 11.5. AlgoTrader UI Alert List

11.1.2. Order Table

The orders will appear in the Order Table in real time. Executed orders are removed from the table after a preset number of seconds which can be configured through the Settings Dialog.
Manual orders can be entered using the order entry form. After entering the order (by specifying security, trade side, order type, quantity, strategy, account and trade type), click the **Submit** button.

To cancel all open orders, click the **Cancel All** button at the top right of the Order Table.

To cancel a specific open order, click the **Cancel** icon besides the order.

To modify an open order, click the **Modify** icon besides the order.

To specify additional order parameters (e.g. time in force and exchange) or place advanced order types (like Algo Orders) click the **Advanced Form** button.
11.1.3. Advanced Order Form

This form lets one enter all types of supported orders, including Algo Orders and allows to use Smart Order Routing feature.

**Figure 11.9. AlgoTrader UI Advanced Order Form**

The first field at top left corner, Order Type, lists all available order types. When Order Type is a simple order type (i.e. Market, Limit, Stop, Stop Limit), then the form allows to specify Time in Force (TIF) and broker/exchange specific parameters, in addition to the Simple Order Form.

For Algo Orders Smart Order Routing is available as well as additional details specific to that type of an Algo Order.

There's also a possibility to use one of pre-defined Order Preferences to fill out the form, they’re listed in the header of Algo Orders properties section. For further details see Section 17.2.1, “Order Preferences”

Below the order type, the Routing section is displayed
Figure 11.10. AlgoTrader UI Advanced Order Form - Routing section, Smart Routing switched off

For Algo Orders you can switch Smart Routing on and off. With Smart Routing switched off you should specify Security, Account, and Exchange to be used.

Figure 11.11. AlgoTrader UI Advanced Order Form - Routing section, Smart Routing switched on

With Smart Routing switched on you can lookup securities with the same symbol and pick the ones that you want Algotrader to pick from.
Figure 11.12. AlgoTrader UI Advanced Order Form - Routing section, Smart Routing disabled

For Simple Orders (Market, Limit, Stop) Smart Routing switch is disabled. In order to use Smart Routing with simple order types you should pick order type ending with "Smart Routing"

Figure 11.13. AlgoTrader UI Advanced Order Form - Routing section, Smart Routing equivalent of Simple Order Type
For details about Smart Order Routing see Section 17.4, “Smart Order Routing”

11.1.4. Algo Order details UI

It is possible to browse and monitor the state of an Algo Order from the web UI. In order to do that, go to the Order tab, and click on the Algo Order icon. A new pop-up window will appear with details of the Algo Order and an additional table with the list of all placed Children orders of the Algo Order (along with details like quantity and state).

![Image](image1)

**Figure 11.14. Execution Algo details icon visible in action column in the Orders table**

![Image](image2)

**Figure 11.15. Execution Algo details modal window with a grid listing all children of a given Execution Algo**

11.1.5. RFQ UI

Algotrader allows you to place Requests For Quotes via the user interface. If one of the configured accounts supports RFQs, the UI will display the RFQ button.

![Image](image3)

**Figure 11.16. RFQ Button displayed in the Order section**

If you press the RFQ button, you can submit a price inquiry.
Figure 11.17. RFQ Entry Form filled with example data

In the first step, complete the inquiry form. You must complete all fields on the form. Select the instrument and the accounts to which you want to send an RFQ (you must select at least one account). Then complete the rest of the fields (Strategy, Side, Quantity and TIF).

Note

Most market makers that support RFQ only allow TIF FOK (fill or kill).

The Submit button sends an RFQ to the selected accounts and takes you to the RFQ execution view.
Figure 11.18. RFQ Responses Window

On the RFQ Execution form, you’ll see offers with their statuses and the time remaining to accept the offer. If you click an **accept** button at the right of a row, an order to execute that quote will be sent out. You can redeem more than one offer.
Once the time expires, you can re-issue the RFQ by clicking the Retry button. The request will be sent only for the selected Market Maker.

### 11.1.6. Transaction Table

The executed trades are listed in the Transactions Table. Since strategies can produce a lot of transactions, only the most recent 50 are shown on startup. All transactions can of course be seen/exported from the database.

#### Figure 11.20. AlgoTrader UI Transaction Table

To manually add a transaction, click the + button at the top right of the Transaction Table.
If a transaction's fee currency is different from the quote currency (e.g. for certain crypto-exchanges), the transaction window will show additional rows to reflect the fees.

Where transaction and fee currencies are the same, the transaction fee will be visible in the fee column of the transaction.

Open positions are listed within the Positions Table.
To close all positions, click the Close All button at the top right of the Position Table.

**Note**

Closing all positions through the UI requires the definition of an order_preference with the name DEFAULT. For further details see Section 17.2.1, "Order Preferences".

The default order preference also includes an account, which means this feature is only usable with one account/adapter.

To close a specific position, click the Close x icon besides the position.

To increase a position by a certain quantity, click the Add + icon besides the position.

To reduce a position by a certain quantity, click the Reduce - icon besides the position.

All of these three actions will pre-fill the order form with appropriate data (security, side, amount in case of close action) and move focus to the form. To send the order please validate the account setting is correct and click Submit.

To open a position’s security chart, click the Chart icon besides the position.

### 11.1.8. Market Data Table

Shows market data in real time.

**Figure 11.24. AlgoTrader UI Market Data Table**

<table>
<thead>
<tr>
<th>Exchange Name</th>
<th>Symbol</th>
<th>Last</th>
<th>Vol Bid</th>
<th>Bid</th>
<th>Ask</th>
<th>Vol Ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMART</td>
<td>HOFT</td>
<td>194.69</td>
<td>1</td>
<td>194.04</td>
<td>196.70</td>
<td>1</td>
</tr>
<tr>
<td>SMART</td>
<td>C</td>
<td>54.75</td>
<td>1</td>
<td>54.05</td>
<td>53.21</td>
<td>1</td>
</tr>
<tr>
<td>SMART</td>
<td>BA</td>
<td>196.01</td>
<td>2</td>
<td>197.01</td>
<td>197.01</td>
<td>1</td>
</tr>
<tr>
<td>SMART</td>
<td>AAPL</td>
<td>255.83</td>
<td>2</td>
<td>253.50 *</td>
<td>256.00</td>
<td>1</td>
</tr>
<tr>
<td>Binance</td>
<td>ETH/USD</td>
<td>235.840000</td>
<td>16.450514</td>
<td>235.840000</td>
<td>235.840000</td>
<td>0.006300000</td>
</tr>
<tr>
<td>Binance</td>
<td>ETC/USD</td>
<td>9.4700000000</td>
<td>0.0468000000</td>
<td>9.4700000000</td>
<td>9.4700000000</td>
<td>0.016800000</td>
</tr>
<tr>
<td>CoinbasePro</td>
<td>BTC/USD</td>
<td>4,470.930000</td>
<td>11.850257</td>
<td>4,470.930000</td>
<td>4,470.930000</td>
<td>2.0197674</td>
</tr>
<tr>
<td>Bitstamp</td>
<td>XRP/BTC</td>
<td>4,056.810000</td>
<td>4,056.810000</td>
<td>0.000000490</td>
<td>0.000000090</td>
<td>12,249.0496971</td>
</tr>
<tr>
<td>Binance</td>
<td>BTC/USD</td>
<td>9.450000000</td>
<td>0.2500000000</td>
<td>9.4500000000</td>
<td>9.4500000000</td>
<td>0.2000000000</td>
</tr>
<tr>
<td>SMART</td>
<td>CA</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IDEALPRO</td>
<td>EUR/USD</td>
<td>1.12337</td>
<td>1.12337</td>
<td>1.12337</td>
<td>1.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>IDEALPRO</td>
<td>GBP/USD</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
To subscribe for a security's market data, click the + button at the top right of the Market Data Table.

**Figure 11.25. AlgoTrader UI Market Data Subscribe**

To unsubscribe to a security's market data, click the **Unsubscribe** icon besides the security.

For any given strategy, it is only possible to unsubscribe from securities the strategy has no position. The unsubscribe icon will only be enabled if the corresponding strategy has no position for the given security.

**Figure 11.26. AlgoTrader UI Market Data Unsubscribe**

The security's chart can be opened by clicking the **Chart** icon besides the security.

**11.1.9. Column Selection and Grouping**

All Tables have a configuration button at the top right of the table. Click the Table Settings option in table's dropdown menu on top right corner to select which columns to show. See the following example for the Transaction Table.

To see all columns of a certain type (e.g. Transaction or Strategy) please click on the expand button.
In every table, any column can be sorted (ascending or descending) by clicking on the column header.

A filter can be applied to any column by clicking at the far right part of the column header. The following image shows the transaction table filtered by a specific symbol.

---

**Figure 11.28. AlgoTrader UI Column Filter**

---

### 11.1.10. CSV Export

All Tables have a Export CSV button at the top right of the table. Clicking the button will initiate the export of all the data visible in given table into a `.csv` file. Exported files will get unique names with table name and the export date.

---

### 11.1.11. Chart Widget

The AlgoTrader UI also comes with the interactive TradingView chart library.

TradingView has regular and advanced chart types and it comes with a massive library of over 100 pre-built technical indicators covering the most popular trading concepts.
The chart widget is useful during strategy development (for initial idea generation, validation, etc.) and for monitoring.

**Note**

TradingView widget can work in two modes. It can either use data provided by TradingView itself, or it can use custom market data adapters which are configured in AlgoTrader.

In case of TradingView market data source there might be slight differences between market data shown in the market data table and the chart widget.

Please, see the [TradingView documentation](https://www.tradingview.com/wiki/Main_Page).

**Figure 11.29. AlgoTrader UI Chart Widget**

To switch market data source between TradingView and custom ones, open settings panel.

---

1 https://www.tradingview.com/wiki/Main_Page
Another feature you get while using your custom historical data source is that AlgoTrader will display your orders and executions on the chart as well. The longer arrows are your orders (aggregated by status) and the shorter are the executions.
If you hover over an order arrow, you will see the current status and details of the orders placed in that bar:

If you hover over an execution arrow, you will see the summary of executions that took place in that bar:
To select particular security to be displayed in the chart, click on the chart icon in the operations column in Market Data or Positions grids.
Figure 11.31. AlgoTrader UI Chart Widget - selecting security

To see the order book chart switch the tab above.
After you switch to the order book tab, you can toggle to the aggregated order book view.

**Figure 11.33. AlgoTrader UI Chart Widget - aggregated order book chart**

The AlgoTrader UI is based on the following technologies/architectures:
• HTML5
• React\textsuperscript{2}
• STOMP\textsuperscript{3} over WebSockets\textsuperscript{4}
• RESTful Web services
• Bootstrap\textsuperscript{5}
• TradingView\textsuperscript{6} chart component
• AGGrid\textsuperscript{7} component

11.1.13. HTML5 Custom Widgets

It is possible to extend the AlgoTrader UI with custom widgets to visualize strategy specific data or let the user interact with strategy specific functionality (e.g. modify parameters or state of a strategy). The following screenshot shows an example of the custom widget in use by the Appendix B, Example Strategy "Box":

\textsuperscript{2} https://reactjs.org/
\textsuperscript{3} https://stomp.github.io/
\textsuperscript{4} http://www.websocket.org/
\textsuperscript{5} https://getbootstrap.com/
\textsuperscript{6} https://www.tradingview.com/
\textsuperscript{7} https://www.ag-grid.com/
HTML5 custom widgets use WebSockets over STOMP to communicate with the strategy.

To integrate the HTML5 custom widget into the main AlgoTrader UI the following items need to be created inside the strategy module (in case you are using the EmbeddedStrategyStarter) or inside the algotrader-core module (in case you are using the ServerStarter). The examples are based on the Appendix B, Example Strategy "Box".

```
/src/main/resources/html5/widgets.json
```

This file is a descriptor for all of widgets used in strategy (it is possible to add multiple widgets for single module). The descriptor in JSON format should be an Array type which contain declarations for each widget:
The Widget descriptor must contain following fields:

**id**
Unique id of the Widget

**title**
Title that will be presented in UI component palette and component itself

**type**
Currently only "iframe" is supported (the Widget will be loaded in its dedicated IFRAME)

**entryPoint**
The URL pointing to the Widget's HTML code (the Widget's code can contain any further imports needed)

/src/main/resources/html5/box.html
This file should contain the HTML code for the custom widget. Individual tags will be referenced by JavaScript code through tag ids.

```html
<div class="widget-container">
  <h3 style="align-self:center;">Box Strategy</h3>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Name</th>
        <th>Value</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>State</td>
        <td id="box_state"></td>
      </tr>
      <tr>
        <td>Units</td>
      </tr>
    </tbody>
  </table>
</div>
```
<td id="box_units"></td>

<tr>
    <td>Upper Target</td>
    <td id="box_upperTarget"></td>
</tr>

<tr>
    <td>Upper Buffer</td>
    <td id="box_upperBuffer"></td>
</tr>

<tr>
    <td>Top</td>
    <td id="box_top"></td>
</tr>

<tr>
    <td>Bottom</td>
    <td id="box_bottom"></td>
</tr>

<tr>
    <td>Lower Buffer</td>
    <td id="box_lowerBuffer"></td>
</tr>

<tr>
    <td>Lower Target</td>
    <td id="box_lowerTarget"></td>
</tr>
</tbody>
</table>

<button id="box_terminate" style="align-self:center;" class="btn btn-danger btn-xs" role="button">TERMINATE TRADE</button>

/src/main/resources/html5/box.js

This file should contain the JavaScript code for the custom widget:

```javascript
$.get("/rest/broker/url/ws", function(wsURI) {
    // connect to stomp
    var ws = new WebSocket(wsURI, "stomp");
    var stompClient = Stomp.over(ws);
    stompClient.connect({}, function(frame) {

        // subscribe for metrics updates
        // {"activemq.retroactive" : true} gets the last metrics event upon subscription
        stompClient.subscribe('/topic/strategy.box.metrics', function(message){
            setValues(JSON.parse(message.body));
        });
    });
});
```
Requests the WebSocket URI via REST call
Connects to STOMP of WebSocket
Subscribes for metrics updates of the strategy which are sent to the topic `strategy.box.metrics`. Setting `{"activemq.retroactive" : true}` will allow the custom widget to get the last metrics event from the `strategy.box.metrics` topic upon subscription.
Populates the content tags with the contents of the metrics events
Sets button actions. Clicking the `terminate` button will send an empty message to the `strategy.box.terminate` topic.

The strategy service class is responsible for sending events to the custom widget and for processing incoming events from the custom widget.

Strategy service classes can send events to the custom widget by using the `JsonTemplate` available inside the strategy service. The following code snippet will send a `box` event to the topic `strategy.box.metrics`:

```java
getJsonTemplate().convertAndSend("strategy.box.metrics", box);
```

Strategy service class methods can be annotated with the `JmsListener` annotation in order to receive incoming events from the custom widget. The following code snippet will attach the `terminateSeries` method to the topic `strategy.box.terminate`:

```java
@JmsListener(destination = "strategy.box.terminate")
public void terminateSeries() {
    ...
}
```

To see the full source code of above examples please see the corresponding source code of the Appendix B, Example Strategy “Box”. Additional HTML5 custom widgets are available inside the example strategies Appendix D, Example Strategy “IPO” and Appendix C, Example Strategy “Pairs Trading”.

### 11.2. Reference Data Manager

The AlgoTrader Reference Data Manager (RDM) is a web based tool accessible via the AlgoTrader UI. It provides a way to manually edit reference data related database tables with user friendly interface and without the need to restart AlgoTrader after making changes. The list of editable tables is as follows:
The Reference Data Manager also allows running the reference data download from the user interface without having to run ReferenceDataStarter separately.

In order to use RDM you need to start AlgoTrader with html5 Spring profile enabled. You can then access RDM through the left-side menu of the AlgoTrader client, menu item Reference data.
If you start AlgoTrader with a reference data Spring profile enabled (see Chapter 20, Reference Data), RDM will allow you to run the reference data download from its interface. After running it, the new securities, security families, etc. will be immediately available to use with AlgoTrader without a restart. The following image shows an example where the reference data download was run with AlgoTrader set up with Bitfinex exchange reference data Spring profile.

The new securities are visible in the Security tab in RDM.
The new securities are available in the AlgoTrader UI.

It is possible to enable a FIX adapter at run time, by activating the relevant Account table entry via RDM.
11.3. Historical Data Manager

The Historical Data Manager (HDM) is a web based tool accessible via the AlgoTrader UI. It provides a way to download, view and edit historical data (see Chapter 19, Historical Data) stored in InfluxDB database if AlgoTrader is configured with. It also has a functionality for exporting and importing the historical data from/to InfluxDB in different CSV file formats. In order to use Historical Data Manager, AlgoTrader needs to be started with html5, influxDB Spring profiles activated and a historical data adapter profile (e.g. iBHistoricalData for Interactive Brokers adapter).

You can access the HDM through the left-side menu of the AlgoTrader client.
In the following screen shot, the view of 1 minute AMZN stock historical data is selected. The data table can be ordered by dateTime in ascending or descending order. After selecting an instrument in the tree menu on the left side of the screen, all instrument's data with the relevant parameters (i.e. adapter type and bar size) is displayed.
The date and time range of displayed data is visible in the **Min. date** and **Max. date** fields at the top of the screen. The date and time range of displayed data may be changed by editing the two fields and clicking the **Set range** button. Note also the **Delete all** button at the top, it removes all data that is currently visible.

After clicking on a row in the historical data table, a window is opened from where an individual measurement may be edited or deleted.

---

**Table:** Date and Time Range

<table>
<thead>
<tr>
<th>Date</th>
<th>Min. Date</th>
<th>Max. Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-04-07</td>
<td>06:00:00</td>
<td>23:59:59</td>
</tr>
</tbody>
</table>

---

**Table:** Historical Data Import

<table>
<thead>
<tr>
<th>Date</th>
<th>Min. Date</th>
<th>Max. Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-04-07</td>
<td>06:00:00</td>
<td>23:59:59</td>
</tr>
</tbody>
</table>

---

**Table:** Add new tick

<table>
<thead>
<tr>
<th>Date</th>
<th>Min. Date</th>
<th>Max. Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-04-07</td>
<td>06:00:00</td>
<td>23:59:59</td>
</tr>
</tbody>
</table>
Clicking the Add new bar button opens the window for entering a new measurement.

Note

The HDM only displays dates of measurements down to seconds, even if they have smaller than a second fractions defined in the InfluxDB store. InfluxDB allows storing measurements with
precision down to nanosecond. Adding a new measurement with a date that is exactly same as an existing measurement (same in nanoseconds), will effectively overwrite the existing one.

The Export into CSV button lets the user export the data currently displayed in the historical data table in CSV file format. The HDM supports several different CSV file formats.
This is an example of a CSV file exported in Excel CSV Type format.

When selecting Historical Data Import in the tree menu on the left, the following screen is displayed. It provides a facility to import CSV files in several formats. Imported data is stored in InfluxDB.

Note that importing consists of 2 phases, the file upload phase where the uploaded file is stored in a temporary file in the operating system where AlgoTrader server runs, and the file processing phase which converts and stores the data in InfluxDB.
When selecting **Historical Data Download** in the tree menu on the left, the following screen is displayed. It provides a facility to download data via the historical data adapter AlgoTrader is currently configured with. Downloaded data is directly stored in InfluxDB. This functionality is only available in HDM when a historical data adapter is configured. The functionality is identical to running a Historical Data Starter (see Section 19.3, "Historical Data Download"). After finishing the historical data download, the last few lines of the AlgoTrader server log is displayed.
Note

An individual historical data download might take a considerable amount of time depending on the date range and granularity (e.g. 1 year worth of 1 second bars).
Performance Monitoring

AlgoTrader provides a sophisticated performance metrics tracking functionality. The main components are meters, registries and Grafana.

- Meters are used to monitor certain performance aspects (metrics) of the application.
- Registries are acting as data pipelines that can be used to store/forward the measurements taken by the meters. Currently Influx, Logging and JMX are the supported solutions.
- Grafana is the supported data visualization solution.

12.1. Metrics Configuration

By default all performance metrics are enabled and to be collected and persisted (if InfluxDB is enabled via the influx profile). See the default config settings and a description on how to use them (conf-metrics.properties):

```
#List of metrics enabled and disable switches delineated by |. +{switch} enables -{switch} disables.
#The switches can be found in the MeterSwitch enum. The order of switches matter, and
#gets evaluated left to right. e.g.:
#  -to enable all but esper: +All|-Esper
#  -this only enables all due to ordering: -Esper|+All
#  -a more complex one: +AlgoTrader|+Platform|-Hazelcast
#{"type":"String","label":"Enable metrics with +, disable with - (if not enabled doesn't need to disable)"}
metrics.filter=+All

#Additional tags (key:value pairs) added to the specified metric, delineated by |.
# Defined such as metrics.tags.{metric}={tag1}:{value1}|{tag2}:{value2}|...
#The metrics can be found in the MeterSwitch enum.
#e.g.:
#metrics.tags.Esper=type:event|source:AT|color:red
#metrics.tags.EventDispatch=type:event|source:AT|color:blue
#metrics.tags.Order=type:business|source:AT|color:blue
#metrics.tags.MarketData=type:business|source:AT|color:orange

#Log level for metrics (only matters if the LOGGING registry is enabled)
#{"type":String,"label":"Log level"}
metrics.log.level=Debug

# List of registries (possible values that can be added INFLUX,JMX,LOGGING)
#{"type":String,"label":"List of registries to send metrics to for further processing, persistence or presentation"}
metrics.registries=INFLUX,JMX,LOGGING
```
To tweak monitoring behavior, configuration parameters can be set/overridden via program arguments, see Section 2.4, “VM Options”. To disable metrics set the metrics.filter parameter to empty (i.e.: `-Dmetrics.filter=`), this can be useful if the hardware resources running AlgoTrader are limited, and we want to minimize memory consumption and maximize performance.

12.2. Grafana Configuration

AlgoTrader provides out of the box dashboards and a data-source configuration for Grafana under the `/algotrader/bootstrap/conf/src/main/resources/grafana/conf` directory. Grafana loads configuration and dashboards on startup, so the mentioned configuration can be copied over to the Grafana installation's `/conf` directory, and after a start/restart dashboards should load. Please note that there are a few requirements for this:

- Grafana\(^1\) is installed on the same host as AlgoTrader
- InfluxDb must be running
- The influxDB spring profile must be enabled, see Section 26.2, “Spring Profiles”
- The INFLUX registry must be enabled, see Section 12.1, “Metrics Configuration”

Please be aware that for specific dashboards to work, that metric has to be enabled on the AlgoTrader side too, otherwise AlgoTrader would not collect/persist that data and Grafana would have no way of reading it from Influx. e.g.: If Esper metrics are not enabled on the AlgoTrader side (see Section 12.1, “Metrics Configuration”), then all he Esper Statements and Engine related dashboards will stop displaying data.

12.3. Metrics, Dashboards, Panels

AlgoTrader has a wide range of metrics that can be monitored. All of them are fed to the enabled registries, see Section 12.1, “Metrics Configuration”.

12.3.1. Environment Metrics

Metrics and dashboards including general CPU, Process, Thread, Memory and JVM monitoring.

---

\(^1\) http://grafana.com
Dashboard Panels:

- **CPU** - Shows the CPU count and current CPU usage (current system load).

- **Threads** - Shows the number of threads and their states. Thread usage statistic that shows how well threads make use of the available resources. High blocked threads may point to locking issues, and similarly a high number of waiting threads could indicate that the CPU cannot keep up with the load or that there are not enough cores to split the work in an efficient manner.

- **GC Pause** - Shows when and for what reason garbage collection pauses happen and how long they last (in nanoseconds).

- **GC** - Displays general garbage collector tracked memory usage. Could indicate the reason why garbage collection happens.

- **Class Loader** - Tracks the number of classes loaded and unloaded.

- **JVM Memory** - Monitors the java virtual machines memory management in terms of memory type (heap, non-heap) and area/generation. This can be used to detect memory leaks, programming inefficiencies, e.g.: a 10x "heap|PS Eden Space" type memory allocation compared to "heap|PS Old Gen" could indicate unnecessary object creations/memory leaks. We can also determine memory boundaries available to the application looking at the maximum values. Co-relating this with other charts can be useful in narrowing down memory management issues in the application.

- **JVM Buffer** - Shows the jvm buffer count and memory usage.

### 12.3.2. Esper Metrics

Esper statement and engine metrics.
Engine Metrics can be seen at the top at all times and have the following attributes as per Esper:

- **inputCount** - Cumulative number of input events since engine initialization time. Input events are defined as events send in via application threads as well as insert into events.
- **inputCountDelta** - Number of input events since last reporting period.
- **scheduleDepth** - Number of outstanding schedules


Statement Metrics can be seen on the bottom panel and can be set to display only selected statements. The shown attributes are the following as per Esper:

- **wallTime** - Statement processing wall time in nanoseconds (based on `System.nanoTime`). This is the most useful metric from esper as it indicates how much time is spent on a specific Esper statement, that can indicate either a bottleneck in the code called by the esper statement, or that the statement itself is too resource heavy, complicated or overloaded.
- **numInput** - Number of input events to the statement.
- **numOutputIStream** - Number of insert stream rows output to listeners or the subscriber, if any.
- **numOutputRStream** - Number of remove stream rows output to listeners or the subscriber, if any.

---

See statement metrics\(^3\).

### 12.3.3. Hazelcast Metrics

Two types of hazelcast metrics are tracked. One is the Hazelcast side general IMap operation performance, as opposed to the Facade metrics that are AlgoTrader's abstraction over Hazelcast, tracking AlgoTrader specific operations.

### 12.3.3.1. Cache Metrics

Provides memory, latency, call-count and size information on Hazelcast caches.

Dashboard Parameters:

- **cache_name** - Selects the cache or caches to be observed/compared with each other.
- **meter** - Selects the aspect of the cache that we want to observer/compare.

This dashboard is useful to monitor Hazelcast cache performance (call-counts, memory-usage, latency). Allows the user to pinpoint slow, overloaded or memory intensive caches that could be a source of performance issues.

### 12.3.3.2. Cache Facade Metrics

Provides actual operation call-count (count), operation-time mean (average), upper (maximum) and sum (cumulative) measurements for each measurement period.

---

This dashboard is similar to Cache Metrics in that it tracks cache usage performance (call-counts, latency), but at a higher level and allows the user to pinpoint specific methods that can be the source of bottlenecks. This is the main metric used to find the ideal performance hot-spots to optimize in the caches.

### 12.3.4. Market Data Metrics

Used to monitor Exchange and internal AlgoTrader latency and event count, measuring how much it takes for specific types of market data (including ticks) to arrive from different market data providers, and how long it takes to propagate them internally. This is not available for all providers as not all venues provide the necessary data (sending-time).
There are two blocks of panels. The upper section has Internal count and latency figures, while the bottom block of panels contains the Exchange figures. The connection between the two lies in the adapter layer, so internal really means Adapters-to-Strategies and exchange means Exchanges-to-Adapters, describing the external (network) and internal path market data takes. These metrics are useful for monitoring the network connection to the exchanges and to detect outages and general responsiveness, to determine the source of adapter issues, whether it is internal/external, load/network/latency related.

Dashboard Internal Panels:

Monitors AlgoTrader internal data paths.

• Count By
• **Adapters and Classes** - Identifies how many of a type (internal data class) of event an adapter (from the exchange) propagates internally (to Strategies) during the last measurement period.

• **Adapters** - The number of events an adapter propagates to strategies, this is an aggregation of the above mentioned data by adapters.

• **Classes** - The number of events grouped by type propagated from all the adapters combined, this is an aggregation of the first panel's data by event type (internal data class).

**Delay By**

• **Adapters and Classes** - Measures the latency of by type (internal data class) of event an adapter propagates internally (to strategies) during the last measurement period.

• **Adapters** - Latency of events an adapter propagates to strategies, this is an aggregation of the above mentioned data by adapters.

• **Classes** - Latency of events grouped by type propagated from all the adapters combined, this is an aggregation of the first panel's data by event type (internal data class).

**Dashboard Exchange Panels:**

Monitors Exchange (network) communication.

• **Count By**

• **Adapters and Classes** - Identifies how many of a type of event (exchange data class) an exchange sends to AlgoTrader (adapters) during the last measurement period.

• **Adapters** - The number of events an exchange sends to AlgoTrader (received by the exchange specific adapter), this is an aggregation of the above mentioned data by adapters/exchanges.

• **Classes** - The number of events grouped by type sent to AlgoTrader (received by the exchange specific adapter) from all the exchanges combined, this is an aggregation of the first panel's data by event type (exchange data class).

• **Delay By**

• **Adapters and Classes** - Measures the latency of by type (exchange data class) of event an exchange sends to AlgoTrader (adapter) during the last measurement period.

• **Adapters** - Latency of events an exchange sent to AlgoTrader (received by the exchange specific adapter), this is an aggregation of the above mentioned data by adapters/exchanges.

• **Classes** - Latency of events grouped by type sent to AlgoTrader (received by the exchange specific adapter) from all the exchanges combined, this is an aggregation of the first panel's data by event type (exchange data class).
12.3.5. Order Metrics

These metrics track exchange and internal AlgoTrader latency and event count, measuring the delay between sending orders and receiving a response/status (ack, fill, reject, cancel) from the target trading venue. This is not available for all providers as not all venues provide the necessary data (sending-time).

**Note:** the numbers on this screenshot are not representative as they have been recorded on a demo account that has some order limitations put in place inflating the numbers.

There are two blocks of panels the upper section has internal count and latency figures, while the bottom block of panels contains the exchange figures. The connection between the two lies in the adapter layer, so Internal really means Adapters-to-Strategies and exchange means Exchanges-to-Adapters, describing the external (network) and internal path market data takes. These metrics are useful for monitoring the network connection to the exchanges and to detect outages and general responsiveness, to determine the source of adapter issues, whether it is internal/external, load/network/latency related.

Dashboard Internal Panels:
Monitors AlgoTrader internal data paths.

- **Count By**
  - **Adapters and Classes** - Identifies how many of a type (internal data class) of order were sent (by strategies) internally (to adapters) during the last measurement period.
  - **Adapters** - The number of orders strategies propagate to an adapter, this is an aggregation of the above mentioned data by adapters.
  - **Classes** - The number of orders grouped by type propagated to all the adapters combined, this is an aggregation of the first panel's data by event type (internal data class).

- **Delay By**
  - **Adapters and Classes** - Measures the latency by type (internal data class) of order sent (by strategies) internally (to adapters) during the last measurement period.
  - **Adapters** - Latency of events an adapter propagates to strategies, this is an aggregation of the above mentioned data by adapters.
  - **Classes** - Latency of events grouped by type propagated from all the adapters combined, this is an aggregation of the first panel's data by event type (internal data class).

**Dashboard Exchange Panels:**

Monitors Exchange (network) communication.

- **Count By**
  - **Adapters and Classes** - Identifies how many of a type of order (exchange data class) an exchange sent to AlgoTrader (adapters) during the last measurement period.
  - **Adapters** - The number of order responses an exchange sends to AlgoTrader (received by the exchange specific adapter), this is an aggregation of the above mentioned data by adapters/exchanges.
  - **Classes** - The number of order responses grouped by type sent to AlgoTrader (received by the exchange specific adapter) from all the exchanges combined, this is an aggregation of the first panel's data by event type.

- **Delay By**
  - **Adapters and Classes** - Measures the latency of a type of order response (SUCCESS, FAIL) an Exchange sent to AlgoTrader (adapter) during the last measurement period.
  - **Adapters** - Latency of order responses an exchange sent to AlgoTrader this is an aggregation of the above mentioned data by adapters/exchanges.
  - **Classes** - Latency of order responses grouped by type sent to AlgoTrader from all the exchanges combined, this is an aggregation of the first panel's data by event type.
12.3.6. Tick-to-Trade

Unlike other metrics this one is not readily available and calculated, and needs some additional math/guesswork to have. All the necessary data to calculate the Tick-to-Trade metric is available as smaller parts of other metrics and panels on the Section 12.3.4, “Market Data Metrics” and Section 12.3.5, “Order Metrics” dashboards. To have a clearer picture lets have a look at how the Tick-to-Trade is build and what components we specifically need from those two dashboards to be able to calculate it. If the whole process is automated via a strategy then the data path is the following:

Market Data Path:

• Exchange ===> (ExternalMD/Tick) ===> Adapter

• Adapter ===> (InternalMD/Tick) ===> Strategy

Order Path:

• Strategy ===> (InternalOrder) ===> Adapter

• Adapter ===> (ExternalOrder) ===> Exchange

Order Status/Response Path:

• Exchange ===> (ExternalStatus/Resp) ===> Adapter

• Adapter ===> (InternalStatus/Resp) ===> Strategy

Adding up the latencies on these paths, gives us an accurate approximation of the Tick-to-Trade metric. Depending on the requirements it can be calculated in a variety of ways, of which the simplest is to take the Delay Average panel values and add them up by path, be aware that this is only an average though and we can have outliers. The way it translates is the following: the first three points directly correspond to the values displayed on the Market Data and Order dashboards, and the last free can be seen as a single path that we cannot separately measure, only as a cumulative network latency value and is displayed on the Order dashboard.

Since the panels contain more specific data history filtered/grouped by exchange, order type and market data type, mirroring the previous data path calculation process with carefully selected data can determine the Tick-to-Trade metric at a certain time for a fixed order type, market data type and exchange if desirable.

In addition to being able to calculate the Tick-to-Trader metric, having the sub-component panels helps tracking/narrowing down and potentially solve the issue when the value is becomes overinflated.

12.3.7. Event Dispatch Metrics

Tracks event types, event consumers/listeners and the combination of them to capture call-count and time-spent on dispatch/execution to detect slow events, consumers, listeners and potential bottlenecks. The panels are broken down by the type/method of dispatch, event type and listener type.
12.3.8. Influx Metrics

A dump of Influx's own usage data that it keeps, useful to track memory usage and general influx performance.
12.3.9. UI Metrics

Tracks page loads, memory consumption and tick, order book and aggregated order book messages that are streamed into the browser instance. Metrics are discriminated by `browserId` - an unique identifier of given browser instance (it's a local storage variable, i.e. it is set uniquely for each URL and browser combination). `BrowserId` can be checked in either local storage variables (`AG_1_BROWSER_ID`) or in requests parameters. Metrics are first aggregated on UI side and then sent periodically (`html5.metricsPushInterval`) to backend and saved to metrics store. So for example if single data point states that the number of order book messages was equal to 100, it means that 100 messages were received by UI in given period.
Risk Management

Using the provided functionality of the system, the following risk metrics can be enforced by AlgoTrader based strategies:

• Fixed and/or Trailing Stop-Loss Limit for each Position
• Minimum Cash-Balance
• Maximum Loss per Position
• Maximum Leverage of the Portfolio
• Maximum Monthly Draw-Down
• Maximum Market Exposure
• etc.

Note
Enforcing these rules requires coding and is not available through configuration only.

13.1. Pre-Trade Checks

AlgoTrader offers a mechanism to inject arbitrary pre-trade checks. These pre-trade checks can for example reject orders in case of violations of:

• Maximum order quantity
• Maximum orders per time period (e.g. per day)
• etc.

To implement pre-trade checks the following steps are needed:

1. Create a client-specific base project which has a maven dependency to the algotrader-core module. This Project will be used to start the application.

2. Inside this base project provide as many implementations of the OrderValidator class as needed.

3. Provide a custom wiring-class inside the base-project that is initializing these OrderValidators.

```java
public interface OrderValidator {
    void validateOrder(Order order) throws OrderValidationException;
}
When AlgoTrader is started, all OrderValidator implementations are injected into the OrderService. Every time an Order is sent or modified (not on cancel) and one of the OrderValidators throws an exception, the Order will be rejected and will not go out to the broker or exchange.
Forex Handling

The System provides full Forex and Currency Exchange Management. FX Rates can be retrieved in real-time. All portfolio figures are calculated based on up-to-date FX-Rates.

FX conversion is provided through the MarketDataService. When subscribing for a non-portfolio currency instrument the system will automatically subscribe the Forex instrument necessary to convert the non-portfolio currency balance (e.g. realized PnL and market value) into the portfolio currency. In some cases multiple Forex/Currency Pairs (based on the same base and quote currency) are available in the system which are traded through different brokers / exchanges.

For those cases a special setting misc.defaultMarketAdapters defines the priority in which market adapters and exchanges are used. The default value is the following:


This is a comma separated list of preferred <exchange>:<adapterType>. Exchange code is taken from the database security.exchange_code value and adapter type is ch.algotrader.enumeration.AdapterType. Optional specific security pairs are also supported, using '@' symbol, e.g. 'BTCUSD@BITF:CNP'. When the platform needs to make a conversion for a given forex, it will iterate over this list and determine the first match. Match means all of the below:

• if currency pair is specified, it should match the symbol of the Forex
• This currency pair is traded on given exchange
• Current adapter type is active, e.g. it's Spring profile is enabled (e.g. if cNPMarketData is active then 'CNP' will be matched for CoinApi securities)

To change the default settings the following property needs to be updated inside conf.properties. Alternatively the properties can be changed via Section 2.4, “VM Options”

# default exchanges/adapter types to use for market data.
SYMBOL@EXCHANGE_CODE:ADAPTER_TYPE=
-Dmisc.defaultMarketAdapters=IDEALPRO:IB

For example, -Dmisc.defaultMarketAdapters=BTCUSD@BITF:CNP,BITF:BFX means that for BTCUSD conversions Bitmex via CoinApi will be used, for all others direct Bitfinex connection will be used (assuming both CoinApi and Bitfinex adapters are running).

14.1. Currency Handling

In most cases securities are attributed in their currency (as defined by Security). Their market value is attributed towards Market Value.
There are however the following exceptions.

### 14.1.1. Futures

- Futures are fully margined, that's why buying a Future does not actually influence cash but only the margin requirement.
- AlgoTrader however treats Futures as regular Securities (i.e. add Future Market Value to System Market Value and deduct Transaction Price from cash)

#### Note

InteractiveBrokers handles Forex slightly different than AlgoTrader:

- IB displays Future Unrealized P/L under Cash.
- To compare AlgoTrader Balances to IB Balances (if there are Future Positions), one has to compare the Net Liquidation Value and not Cash / Market Value individually.

### 14.1.2. Forex

- Forex (e.g. EUR.USD) consists of the Base Currency (e.g. EUR) and Quote Currency (e.g. USD)
- In Balances Forex are attributed towards Cash (and not system Market Value) in the Base Currency
- The gross value of a transaction is booked in the Quote Currency, whereas the commission is booked in the Base Currency.

#### Note

The TWS Trades Window displays commissions in Trade Currency, but IB Flex Reports displays commissions correctly in the Base Currency.

### 14.1.3. Currency Attribution

The following table describes Currency Attribution of Positions. The logic is implemented by `Position.getAttribution()`

#### Table 14.1. Position Currency Attribution

<table>
<thead>
<tr>
<th></th>
<th>General Security</th>
<th>Forex</th>
<th>Future on Forex</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributed to</td>
<td>Market Value</td>
<td>Cash</td>
<td>Market Value</td>
</tr>
</tbody>
</table>
The following table describes Currency Attribution of Transactions. The logic is implemented by `Transaction.getAttributions()`

### Table 14.2. Transaction Currency Attribution

<table>
<thead>
<tr>
<th>General Security</th>
<th>Forex</th>
<th>Future on Forex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency</td>
<td>Currency of Security</td>
<td>Base of Forex</td>
</tr>
<tr>
<td>Amount</td>
<td>quantity * contract size * price</td>
<td>quantity</td>
</tr>
</tbody>
</table>

14.2. Forex-Hedging

The system provides automatic Forex-Hedging by the Service `ch.algotrader.service.ForexService`. This service will maintain multiple FX Positions to hedge all non base currency balances. For actual Forex-Hedging the following two options exist

14.2.1. Exchange vs. Margin Trading

FX brokers and crypto exchanges support exchange trading and/or margin trading.

Exchange trading allows you to exchange/convert one crypto/fiat amount into another (e.g. converting USD to BTC). Using exchange trading short trades are not possible.

Margin trading allows you to have long and short positions on any pair (e.g. long BTC/USD or short ETH/BTC).

The `SimpleOrder` property `exchangeOrder` (which is false per default) drives which account you are trading against:

- `exchangeOrder = true` means use the exchange account
- `exchangeOrder = false` means use the margin account

---

**Note**

Note that every broker / exchange differs on how to submit exchange account vs. margin account orders.

Some require you to use different securities (e.g. Bitflyer), others different order types or price types.
Margin positions can be useful also for Hedging purposes. Depending on the current exposure of a non-base currency equity portfolio, an offsetting margin position can be maintained to result in a zero non-base currency exposure.

### 14.2.2. FX Future

The second option for FX Hedging is by means of FX Futures which is a subclass of Future.

For some FX security families there are multiple entries available with different contract sizes (e.g. 12'500, 62'500 and 125'000 for EUR.USD). Because of this the `hedgingFamily` has to be defined as a Property on the EUR.USD subscription.

Since FX Futures expire, the Hedging Position has to be rolled before Expiration, which is also done automatically based on configured parameters.
Options & Futures

15.1. Expiration

Both Options and Futures implement the interface `ExpirableI` which has a property `expiration` that represents the expiration date.

For hints on how to roll futures and options, see Section 4.3.1.8, “Rolling of Futures and Options”.

In addition to the `expiration` date (sometimes also called `lastTradingDay`), Futures also have the following two fields:

- `firstNoticeDay`: this is the day on which the buyer of a futures contract can be called upon by the exchange to take delivery
- `maturityMonthYear`. The contract month & year of a future

Note

`expiration` and `maturityMonthYear` do not necessarily need to be within the same month. e.g. `maturityMonthYear: 2015-12, expiration: 2015-11-30`

The following table contains references on how these three fields are used by different market data providers

<table>
<thead>
<tr>
<th>Field</th>
<th>InteractiveBrokers</th>
<th>Bloomberg</th>
</tr>
</thead>
<tbody>
<tr>
<td>maturityMonthYear</td>
<td><code>m_maturityMonthYear</code></td>
<td><code>FUT_MONTH_YR</code></td>
</tr>
<tr>
<td>firstNoticeDay</td>
<td><code>not available</code></td>
<td><code>FUT_NOTICE_FIRST</code></td>
</tr>
<tr>
<td>expiration</td>
<td><code>m_expiry</code></td>
<td><code>LAST_TRADEABLE_DT</code></td>
</tr>
</tbody>
</table>

For an detailed explanation between `lastTradingDay` and `firstNoticeDay` please visit this page on futures expiration

Note

IB also shows an expiration date in TWS which is either the same day or the next day after the last trading day

15.2. Leverage & Exposure

AlgoTrader uses the following definitions for Leverage & Exposure:

---

1 [http://www.futurestradingpedia.com/futures_expiration.htm](http://www.futurestradingpedia.com/futures_expiration.htm)
Table 15.1. Leverage & Exposure

<table>
<thead>
<tr>
<th>Term</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Future Leverage</td>
<td>1</td>
</tr>
<tr>
<td>Option Leverage</td>
<td>( \frac{S}{CV} )</td>
</tr>
<tr>
<td>Option Leverage delta-adjusted</td>
<td>( \frac{S*D}{CV} )</td>
</tr>
<tr>
<td>Nominal Exposure (Position Market Value)</td>
<td>( Q<em>CS</em>CV )</td>
</tr>
<tr>
<td>Notional Exposure</td>
<td>( PositionMarketValue*PositionLeverage )</td>
</tr>
<tr>
<td>Notional Exposure delta-adjusted</td>
<td>( C<em>CS</em>S*D )</td>
</tr>
<tr>
<td>Portfolio Exposure</td>
<td>( \frac{\sum PositionExposure}{NetLiqValue} )</td>
</tr>
</tbody>
</table>

**Figure 15.1. Leverage & Exposure**

- **CS**
  - Contract Size
- **CV**
  - Current Value
- **S**
  - Underlying Spot price
- **D**
  - Delta
- **Q**
  - Quantity

**Note**

AlgoTrader uses the Delta-adjusted Option Leverage as Option Leverage.

### 15.3. Symbol, ISIN & RIC

The two classes `FutureSymbol` and `OptionSymbol` provide static methods for generating the Symbol, ISIN and RIC for Futures and Options. Both Symbol and ISIN use the property `symbolRoot` of the `SecurityFamily`. The RIC on the other side uses the property `ricRoot`.

### 15.4. Delta Hedging

Automatic Delta Hedging is provided by the Service `ch.algotrader.service.OptionService`. The method `hedgeDelta` will calculate the delta adjusted market value for all securities based on one particular underlying. The method will subsequently build up a hedging position based on the nearest Future of the same underlying.
Since some underlying have more than one Futures Chain based on them, the property `hedgingFamily` needs to be defined on the Subscription of the underlying. In addition the method `hedgeDelta` can also be invoked with the `ServerManagementService`.

### 15.5. Option & Future Chain Download

The `ReferenceDataService` can be used to download all currently traded and past Options and/or Futures into the database, see Chapter 20, Reference Data.

### 15.6. Option Greeks

The following Option Greeks are available through the Class `OptionUtil`:

- option price (through Black-and-Scholes)
- implied volatility (through Black-and-Scholes, Newton Rapson Method and SABR Surface)
- intrinsic value
- delta
- vega
- theta
- forward price
- moneyness
- strike by delta

### 15.7. Option Pricing Engine

The system provides a sophisticated option pricing engine which is developed around the SABR volatility Model.

Based on historical Volatility at different Moneyness levels (e.g. ATM, ATM +10%, ATM +20%, ATM -10% & ATM-20%) or Delta levels (e.g. 50%, 35%, 75%) volatility parameters are calculated (=calibration) and used for option pricing.

#### 15.7.1. SABR Calibration

The `OptionService` is responsible for SABR calibration. The calibration process happens for one specific expiration and takes an array of strikes with their corresponding array of volatilities. The calibration process returns a `SABRSmile ValueObject`, which basically contains the three parameters `rho, volVol` and `alpha` (in addition to the time-to-expiration and at-the-money volatility). The actual calibration happens through the class `SABR`.

SABR Calibration can be done either by actual Option Prices or directly by the Implied Volatility. Also there are methods to do a SABR Calibration just for one expiration (returning one SABR Smile Value Object) or
for an entire Volatility Surface (returning a SABRSurface ValueObject which consist of multiple SABRSmile ValueObjects)

The SABR Calibration depends on different Implied Volatilities (a subclass of Security) being defined in the database. A Implied Volatility needs to define either a moneyness or a delta (in addition to Duration and Option Type).

15.7.2. Option Pricing

Based on the SABR Calibration the actual option pricing takes place. This is handled through the class OptionUtil with the method getImpliedVolatilitySABR. This method takes SABRSurface parameter. The actual option pricing happens in two steps:

1. For all available expirations a volatility is calculated for the requested strike
2. Using spline interpolation the volatility for the requested expiration is calculated

15.7.3. References

- Hedging under SABR Model, Refined risk management under the SABR model\(^2\)
- Extensions of the SABR model for equity options\(^3\)

15.8. OTC Options

Since OTC Options do not have a predefined chain definition, the OptionService contains a method createOTCOption to create an OTC option based on the specified expirationDate, strike and type.

\(^3\) [https://essay.utwente.nl/59228/1/scriptie_I_Khomasuridze.pdf](https://essay.utwente.nl/59228/1/scriptie_I_Khomasuridze.pdf)
Chapter 16.

Broker/Exchange Interfaces

The System provides generic interface functions to connect AlgoTrader to different brokers / exchanges.

The following broker / exchange specific interfaces are currently available:

- Fix Interfaces
  - B2C2
  - Bitstamp
  - Blockfills
  - Bloomberg Tradebook
  - Coinbase Pro
  - CryptoFinance
  - Currenex
  - Deribit
  - DukasCopy
  - Enigma
  - Exante
  - EzeSoft/RealTick
  - Fortex
  - FXCM
  - Gemini
  - InteractiveBrokers
  - JP Morgan
  - LMAX Digital
  - LMAX Global
  - Nexus Prime
  - One Zero
  - PrimeXM
- SocGen
- Trading Technologies (TT)
- UBS
- B2C2
- Bitstamp
- Coinbase Pro
- Tilde
- Native Interfaces
  - Bloomberg
  - Interactive Brokers
- REST/WebSocket Adapters
  - Binance
  - Bitfinex
  - Bitflyer
  - BitHumb Pro
  - BitMEX
  - Blttrnex
  - CryptoFinance
  - Enigma
  - eToroX
  - Huobi Spot
  - Kraken Spot
  - OKEx
  - OKCoin
  - Refinitiv Elektron (Reuters)

AlgoTrader uses QuickFix/J\(^1\) for all fix interfaces.

\(^1\) https://www.quickfixj.org/
For further details on Broker/Exchange adapters please see *Chapter 23, Adapters*
Order Management

17.1. Order Validation

Before sending an Order, it is advised to call the validate method on the order. This will validate the order regarding limits, amount, quantity, etc. In case validation fails an Exception will be thrown and the order can be modified.

Note

The validate method will be called (again) inside the sendOrder method, in case the validation fails an Exception will be thrown.

In addition to the validate method all configured Section 13.1, “Pre-Trade Checks” will be invoked. In case any pre-trade check is breached the order will be rejected.

17.2. Place Order

The method sendOrder of the OrderService is responsible for placing Orders. This method takes an Order Entity or Order Value Object as parameter.

Sending an order using and Order Entity looks like this:

```java
MarketOrder order = MarketOrder.Factory.newInstance();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
getOrderService().sendOrder(order);
```

The associated Entities (i.e. strategy, account and security) can be retrieved via the LookupService.

Sending an order using Order Value Objects looks like this:

```java
MarketOrderVO order = MarketOrderVOBuilder.create()
.setStrategyId(strategyId)
.setAccountId(accountId)
.setSecurityId(securityId)
.setQuantity(orderQuantity)
.setSide(Side.BUY)
.build();
```
Creating and sending an AlgoOrder looks like this:

```java
SlicingOrder order = new SlicingOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setMinQuantity(BigDecimal.valueOf(10));
order.setMaxQuantity(BigDecimal.valueOf(100));
order.setMinVolPct(0.01);
order.setMaxVolPct(0.1);
order.setMinDuration(1.0);
order.setMaxDuration(5.0);
order.setMinDelay(1.0);
order.setMaxDelay(5.0);
getOrderService().sendOrder(order);
```

Sending an algo order using Order Value Objects looks like this:

```java
SlicingOrderVO order = SlicingOrderVOBuilder.create()
  .setStrategyId(strategyId)
  .setAccountId(accountId)
  .setSecurityId(securityId)
  .setQuantity(orderQuantity)
  .setSide(Side.BUY)
  .setMinQuantity(BigDecimal.valueOf(10))
  .setMaxQuantity(BigDecimal.valueOf(100))
  .setMinVolPct(0.01)
  .setMaxVolPct(0.1)
  .setMinDuration(1.0)
  .setMaxDuration(5.0)
  .setMinDelay(1.0)
  .setMaxDelay(5.0)
  .build();
getOrderService().sendOrder(order);
```

The broker / exchange specific SimpleOrderExecService will create the broker / exchange specific order, assign an intId if none has been assigned yet and send the order to the broker / exchange.
After sending the Order to the broker / exchange, the order object is propagated to the AlgoTrader Server Esper service instance (running inside the AlgoTrader Server) as well as to the Esper service instance of the corresponding strategy (where potential actions like cancel order or modify order can be executed).

Open orders are kept in an internal order book until their full execution or cancellation. Completed orders remain in the book and are accessible through OrderService until evicted. AlgoTrader evicts completed orders at 24:00 local time daily by default. One can also manually evict orders by calling OrderService#evictExecutedOrders() method.

The actual exchange an Order is sent to will be retrieved from the associated Security/SecurityFamily. Alternatively it is possible to assign an Exchange to an Order Entity or Order Value Object directly.

17.2.1. Order Preferences

As AlgoOrders typically have a lot of parameters (e.g. minQuantity, maxQuantity for the SlicingOrder) it is possible to save a set of settings using the OrderPreference Entity. The following OrderPreference SLICING is contained within the db-samples (/algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql):

```
INSERT INTO `order_preference` (`ID`, `NAME`, `ORDER_TYPE`, `PROPERTIES`, `DEFAULT_ACCOUNT_FK`, `VERSION`) VALUES
(202,'SLICING','SLICING','{"minDelay":2,"maxDelay":3,"maxVolPct":1,"minDuration":1,"maxQuantity":20,"minVolPct":0.2,"maxDuration":2}',100,0);
```

The OrderPreference SLICING defines default settings for the Slicing Order. Through the column DEFAULT_ACCOUNT_FK it is also possible to define a default account for the AlgoOrder. With this information available in the database an order can now be created as follows:

```
Order order = getOrderService().createOrderByOrderPreference("SLICING");
order.setStrategy(strategy);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
getOrderService().sendOrder(order);
```

17.2.2. Trade Suggestions

In addition to the OrderService#sendOrder method there is a OrderService#suggestOrder method, which will not send out an order to the broker / exchange but instead create an Email with a Trade Suggestion sent to the registered Email addresses. This allows for manual confirmation of automatically generated Signals.
17.2.3. Order Properties

Even though the Order Entity already contains commonly used properties like side, quantity, time-in-force, etc., it is possible to attach additional arbitrary properties to an Order. Order properties have a name, a value and a type.

Order properties of type INTERNAL are kept inside the system and are not propagated to external brokers.

If the type of an Order property is FIX it is assigned to an outgoing Fix order as an additional fix tag. It is therefore mandatory that the name of the order property is a number (representing the fix tag).

If the type of an Order property is FIX_LITERAL it is parsed and assigned to an outgoing Fix order as a list of fix tags. It is therefore mandatory that the value of the order property is a string consisting of comma delimited entries tag=value.

If the type of an Order property is IB the system will try to find an IB order field corresponding to the IB order property name. In case no matching field is found the order property is added as an AlgoParams.

An internal order property can be added to an order as follows:

```java
order.addOrderProperty("test", "XYZ", OrderPropertyType.INTERNAL);
```

An Fix order property can be added to an order as follows:

```java
order.addOrderProperty("4000", "XYZ", OrderPropertyType.FIX);
```

An IB order property can be added to an order as follows:

```java
order.addOrderProperty("faGroup", "group1", OrderPropertyType.IB);
```

Any object can be attached to an order as a property value using addProperty(String name, Object value) method, it can be done as follows:

```java
order.addProperty("myProperty", "propertyValue");
```

Order Property can be removed from order as follows:

```java
order.removeProperty("name");
```

---

1 http://interactivebrokers.github.io/tws-api/classIBApi_1_1Order.html
2 http://interactivebrokers.github.io/tws-api/classIBApi_1_1Order.html
The method `getOrderProperties` cannot be used to manage order properties it's returning unmodifiable map of `OrderProperty` objects excluding all other properties provided to order, to remove or update properties it's needed to use methods above.

**Note**

`OrderProperties` are only supported on Order Entities but not on Order value objects (VOs).

In order to attach fix properties to the VO when sending or modifying an order you can use convenience methods on `OrderService`: `sendOrderWithFixProperties`, `modifyOrderWithFixProperties` as compared to the usual `sendOrder` and `modifyOrder` methods.

### 17.3. Order requests and confirmations

Apart from sending a new Order, there are several other Order operations that you can request from `OrderService`:

- **Send Order** - send a new Order
- **Modify Order** - modifies already placed Order. Only one request for Order modification can be active at a time. Also, modification request will be rejected if Order was requested to be cancelled already.
- **Cancel Order** - cancels already placed Order.

**Order request confirmations inside Strategy**

You can be explicitly notified inside the Strategy about confirmations or rejections for each request that you placed via Order Service. This functionality can be considered an alternative approach to being notified about OrderStatus changes (this is more extensive solution than OrderStatus notification - as OrderStatus information won't give you information about Order modification requests being rejected for example). To subscribe for Order Request Statuses, use listener below in your `StrategyService` implementation:

```java
@override
public void onOrderRequestStatus(OrderRequestStatusEventVO event) {
}
```
17.4. Smart Order Routing

When placing an order to the system you can specify multiple venues and make Algotrader pick the best one before placing the order to one of them. The main factor affecting that affects candidates ranking is estimated order cost, trading session aliveness, estimated fees and liquidity. Smart Routing is available for all Algo Orders. Simple Orders (market, limit, stop, stop/limit) have their Smart equivalents too.

Note

- Smart Routing is only supported for Stock and Forex spot securities
- You can configure the estimated fees that are used by smart orders in the security or security_family data by setting one or more of
  - EXECUTION_COMMISSION_PER_ORDER
  - EXECUTION_COMMISSION_PER_CONTRACT
  - EXECUTION_COMMISSION_IN_PERCENT

The fee computation logic is as follows:

Execution commission = commissionPerOrder + commissionPerContract * absOrderQty + commissionInPercent * absOrderQty * price

Alternatively, you can set the estimated fees by security on the order by using the expectedFees field on routing target (see the JSON/Java examples below)

- Smart limit and stop orders are managed/monitored by our system until they are triggered by market data events, at which point they are sent to a broker/exchange. This also means you can place e.g. a smart stop order including an exchange that does not natively support stop orders
- All smart orders are algos, which are ended when AlgoTrader is stopped. This is also true for "simple" smart orders at the moment

In order to place a smart routing order using the API, you should populate the candidates parameter with a list of potential targets Target consists of security, account and exchange

Example JSON payload to send a Smart-Routed order via REST API:

```json
{
    "@class": "ch.algotrader.entity.trade.algo.SmartMarketOrderVO",
    "side": "BUY",
    "strategyId": 2,
    ...
}
```
"exchangeOrder": true,
"quantity": "1",
"tif": "GTC",
"routingCandidates": [
  {
    "securityId": 911,
    "accountId": 200,
    "exchangeId": 200
  },
  {
    "securityId": 903,
    "accountId": 211,
    "exchangeId": 210
  },
  {
    "securityId": 898,
    "accountId": 202,
    "exchangeId": 202,
    "expectedFees": 1.23
  }
]
}

Example Java code:

```java
SmartMarketOrderVO order = SmartMarketOrderVOBuilder.create()
  .setSide(Side.BUY)
  .setStrategyId(2)
  .setExchangeOrder(true)
  .setQuantity(new BigDecimal("1"))
  .setTif(TIF.GTC)
  .setRoutingCandidates(Arrays.asList(
    new RoutingTargetVO(911, 200, 200),
    new RoutingTargetVO(903, 211, 210),
    new RoutingTargetVO(898, 202, 202, new BigDecimal("1.23"))
  ))
  .build();

getOrderService().sendOrder(order);
```

**17.4.1. Additional Smart Order Configurations (Mostly Crypto-Relevant)**

Below configurations currently only apply to simple smart orders.
• Since crypto-exchanges are fairly diverse in what time in force options they support, choosing one option might exclude a lot of exchanges, which do not support it.

We have a configuration on exchange adapter level that allows you to default an order to a different TIF if the one on the order is not supported, e.g.

```plaintext
xyz.supportedTifs=FOK,IOC # (comma-separated list) optional, defaults to all TIF values
xyz.defaultTif=FOK # optional, defaults to GTC
```

This behavior is disabled by default. If you want to make use of the TIF conversion, you need to set property `misc.algoOrderFallbackToDefaultTif` to `true`.

• Some exchanges do not support market orders.

You can configure them to execute market orders as limit orders, at an offset of the current market price, e.g.

```plaintext
xyz.supportedOrderTypes=LIMIT # Exchange xyz does not support MARKET
xyz.placeMarketAsLimit=true
xyz.placeMarketAsLimitOffset=0.01 # (percent) offset from current market price (best bid/ask) at which the limit order will be placed.
```

### 17.5. Order Status

The current status of an order is represented by one or many `OrderStatus` events. Whenever order status events are received back from the broker / exchange, `OrderStatus` events are created and propagated to the AlgoTrader Server Esper service Instance and subsequently to the strategy that initiated the order.

Permitted Order Status transitions are depicted by the following state machine diagram.
If an order does not receive either an Acknowledgment or Fill within a configurable time period (default: 1 sec) after sending the Order, an Exception is thrown, as there might be a problem with the broker / exchange connection. This is enforced by the `NOTIFY_MISSING_ORDER_REPLY` statement which can be turned off by changing the following property inside `conf-core.properties`. Alternatively the properties can be changed via Section 2.4, “VM Options”.

```java
# notify in case there is no reply on an order
statement.notifyMissingOrderReply = false
```

Once all fills corresponding to an Order are fully persisted an `OrderCompletionVO` event is generated and propagated to the Strategy. By the time an `OrderStatus` event is received by the strategy, corresponding database activity might not have been completed yet. However by the time an `OrderCompletionVO` event is received, it is guaranteed that all database activity has been completed. It is therefore safe to invoke any sort of the database lookup at this time. For further details see Section 8.4.8, “Callbacks”.

### 17.6. Receive Fills

Whenever fills are received back from the broker / exchange, Fill events are created and propagated to the AlgoTrader Server Esper service Instance and subsequently to the strategy that initiated the order.
The Fill events trigger the creation of a Transaction object (a persistent Record in the database). In addition the Fill and corresponding Transactions are also propagated to the strategy, where actions can be taken upon.

**Note**

Fills and Transactions are separated from each other for the following reason. A Fill contains all the information received from the broker / exchange (and a reference to the Order). Whereas a Transaction contains all the information related to accounting (i.e. references to position and strategy). In addition to Transactions related to Fills, there are Transactions that are independent of Fills (i.e. Deposits, Withdrawals, Interest, etc.).

### 17.7. Handling of Fees and Commissions

AlgoTrader supports handling of the following three types of fees and commissions:

- Exchange Fees
- Execution Commissions (typically charged by executing brokers)
- Clearing Commissions (typically charged by clearing brokers)

All three types are available in the database table transaction.

Most adapter for traditional asset classes (e.g. equities, forex and derivatives) do not provide fee information on execution messages. Some crypto exchanges provide fee information. If the fees are in the currency of the transaction, they are stored in the fee attribute of the transaction. In case the fees are charged in another currency (for example Binance charges in its own currency - BNB), a new transaction is created with transactionType = FEES, quantity = -1 for fees paid (and quantity = -1 for fees received, e.g. maker rebates), price = fee value and currency = fee currency.

For the other adapters, AlgoTrader uses the internal Execution Model (see Section 5.1, “Exchange Simulator”) to assign fees and commissions based on configuration (e.g. commission-per-contract).

### 17.8. Examples of Orders and Executions

The following sections show a few example orders and their corresponding executions/fills. All examples start from a "clean" database with empty transaction, cash_balance and position tables.

#### 17.8.1. Margin Order with Fee in Quote Currency

Margin Order: BUY 0.002 BTC/USD @ Bitfinex

Execution: 0.002 @ 4363.20, gross value: 8.7264, Fees: 0.01745280 USD

This will result in the following values:

Transactions:
17.8.2. Exchange Order with Fee in Base Currency

Exchange Order: BUY 0.003 BTC/USD @ Bitfinex

Execution: 0.003 @ 4374.23439227, gross value: 13.12270318, Fees: 0.000006 BTC

This will result in the following values:

Transactions:

<table>
<thead>
<tr>
<th>PRICE</th>
<th>QUANTITY</th>
<th>FEE</th>
<th>CURRENCY</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.12270318</td>
<td>-1</td>
<td>0.000006</td>
<td>USD</td>
<td>EXCHANGE_DEBIT</td>
</tr>
<tr>
<td>0.003</td>
<td>1</td>
<td></td>
<td>BTC</td>
<td>EXCHANGE_CREDIT</td>
</tr>
</tbody>
</table>

Cash Balances:

<table>
<thead>
<tr>
<th>CURRENCY</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>USD</td>
<td>-13.12270318</td>
</tr>
<tr>
<td>BTC</td>
<td>0.002994</td>
</tr>
</tbody>
</table>

No positions as this is an exchange order

17.8.3. Exchange Order with Fee in Alternate Currency

Exchange Order: BUY 0.003 BTC/USDT @ Binance

Execution: 0.003 @ 4395.51, gross value: 13.18653, Fees: 0.001796 BNB

This will result in the following values:

Transactions:

<table>
<thead>
<tr>
<th>QUANTITY</th>
<th>PRICE</th>
<th>CURRENCY</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Internal Order Id Format

**intId** is the internally assigned order identifier whereas **extId** is the id assigned by the external broker / exchange.

In general the internal order identifier has the following format:

```
<session_qualifier><id>.<version>
```

**Example:** `lmax1.1`

- **session_qualifier:** each Fix session has a unique session qualifier
- **id:** an integer which is auto-incremented per session. For Fix, the last id is retrieved from the order table during start up
- **version:** The number of modifications that took place on the Order, starting with 0 when the order is first submitted.

By default AlgoTrader automatically assigns an **IntId** value to all outgoing orders. Open and executed orders can be identified and looked up by their **IntId**.

Especially when using a *Section 8.4.8.2, “Trade callback”* it is necessary to generate and assign an **IntId** value to the order prior to submitting it to the order service. The `OrderService#getNextOrderId()` method can be used to generate a unique **IntId** value per session associated with an Account record.

```java
String orderId = getOrderService().getNextOrderId(order.getClass(), accountId);
order.setIntId(orderId);
engine.addTradeCallback(Collections.singleton(orderId), (strategyName, orderStati) -> {
    ...
```
getOrderService().sendOrders(orders);

Note
Please note that care must be taken when using OrderService#getNextOrderId() with the IB order service. The IB native interface expects orders to be sent with their order ids in ascending order. The Class IBOrderIdSynchronizer is responsible to make sure order ids are actually in ascending order. In case an order id is skipped the IBOrderIdSynchronizer will wait for up to maxOrderSyncTime milliseconds for the order with the correct order id to arrive.

17.10. Symbology

In the electronic trading domain there are different ways to identify a security, some of which are:

- Symbol
- Bloomberg ID
- For options: underlying, expiration, strike & type
- etc.

Different Brokers employ different types of Symbology to identify a security. For this purpose AlgoTrader provides the notion of SymbologyResolver which is responsible for assigning appropriate information to outgoing broker / exchange communication. These SymbologyResolvers can be extended on a per adapter basis.
Market Data

AlgoTrader provides Market Data Interfaces with the following market data providers:

- Fix Interfaces
  - B2C2
  - Bitstamp (L2 order book)
  - Blockfills (L2 order book)
  - CryptoFinance
  - Currenex (top of book)
  - Deribit (L2 order book)
  - DukasCopy (L2 order book)
  - Enigma
  - Exante (L2 order book)
  - Fortex (top of book)
  - FXCM (top of book)
  - Gemini (L2 order book)
  - Interactive Brokers (L2 order book)
  - Intrinio (dividends data only)
  - LMAX Global (L2 order book)
  - LMAX Digital (L2 order book)
  - Nexus Prime (L2 order book)
  - One Zero (top of book)
  - PrimeXM (L2 order book)
  - Trading Technologies (L2 order book)
- Native Interfaces
  - Bloomberg (top of book)
  - Interactive Brokers (L2 order book)
• QuantHouse (top of book)

• REST/WebSocket Interfaces
  • Binance (L2 order book)
  • Bitfinex (L2 order book)
  • Bitflyer (L2 order book)
  • BitHumb Pro (L2 order book)
  • BitMEX (L2 order book)
  • Bittrex (L2 order book)
  • CoinAPI (L2 order book)
  • Coinbase Pro (L2 order book)
  • eToroX (L2 order book)
  • Huobi Spot (L2 order book)
  • Kraken Spot (L2 order book)
  • OKEx (L2 order book)
  • OKCoin (L2 order book)
  • Refinitiv Elektron (top of book)

AlgoTrader allows having multiple market data interfaces active at the same time so market data received from different market data providers can be compared in real-time.

To enable either of those Market Data Interfaces the following two steps have to be executed:

1. The correct Spring Profile has to be activated according to Section 26.2, “Spring Profiles”

2. For Bloomberg market-data-bb and for InteractiveBrokers market-data-ib has to be added to the VM argument server-engine.init when running the AlgoTrader server.

Market Data Events itself are available in different types:

• BarVO Open-High-Low-Close Price Bars, also containing volumes and volume weighted average prices

• TickVO: The top of the order book at a particular point in time, containing information like last price, last time, bid, ask, volume, etc.

• QuoteVO: Its subclasses represent the current best bid and offer BidVO and AskVO
• **TradeVO**: An actual order that was executed on the market, containing information like last price, last size and volume

• **GenericTickVO**: Represents additional price information made available by market data providers (e.g. open price, close price, vwap price)

• **OrderBookVO**: A snapshot of the full order book for a security from the trading venue at a particular moment in time

• **AggregatedOrderBookVO**: Contains price levels of the order books from multiple venues combined by a symbol

As the following diagram shows, market data providers deliver Price Events (**TradeVO**, **BidVO & AskVO**) or individual **TickVO** Fields.

In back testing AlgoTrader supports both Ticks or Bars. In both live trading and back testing Price events and Tick events can be aggregated into Bar events

**Figure 18.1. Market Data Event Types**

Inside each strategy the **MarketDataCacheService** keeps a copy of the last **MarketDataEvent** for each Security. For further details see Section 7.2.17, “Market Data Cache Service”.

### 18.1. Creation of Bars based on Ticks

Through the live data recording feature (see Section 19.2, “Live Data Recording”) tick data can automatically be aggregated into bar-data.

In addition, it is possible to aggregate ticks into bars through Esper code in both Simulation and Live Trading by using an Esper `time_batch` window:

```sql
select
```
18.2. Creation of Bars based on Bars

You might need to have more than one aggregation level of bars in your strategy. For this you can initially create the smallest bar size you require according to Section 19.2, “Live Data Recording” and then aggregate these bars to get larger bars.

There are several ways to do this:

- Using Esper
- Using Java

18.2.1. Esper Bar Aggregation

Aggregating 1 minute bars into 10 minute bars:

```sql
first(last) as open,
max(last) as high,
min(last) as low,
last(last) as close
from
TickVO.win:time_batch(1 min)
group by
securityId;
```

and `expr_batch` for constant volume bars:

```sql
select
max(last) as high,
min(last) as low,
first(last) as open,
last(last) as close
from
TickVO.win:expr_batch(sum(volume) > 1000)
group by
securityId;
```

**Warning**

Some market data providers (e.g. Interactive Brokers) will only provide market data snapshots at regular time intervals. This can cause deviations in the bar high and low price.

Due to clock differences between the local machine and the market data provider’s servers there might be slight deviations in the bar open and close price. It is therefore important to enable system clock synchronization on the server where AlgoTrader is installed.
@Name('AGGREGATE_BAR_TO_BAR')
insert into
   BarVO
select
   first(bar.dateTime),
   last(bar.adapterType),
   last(bar.securityId),
   Duration.valueOf('MIN_10'),
   first(bar.open),
   max(bar.high),
   min(bar.low),
   last(bar.close),
   sum(bar.vol)
from
   BarVO(barSize = Duration.valueOf('MIN_1')).win:time_batch(10 min, 0L, "FORCE_UPDATE")
as bar
  group by
   bar.securityId,
   bar.adapterType;

18.2.2. Java Bar Aggregation

Java Bar to Bar Aggregation can be performed using the AlgoTrader BarToBarAggregation. Inside strategies an aggregation to 10 Minute Bars looks like this.

private BarToBarAggregation aggregation = new BarToBarAggregation(Duration.MIN_10, c -> {
   // action will be performed on each newly produced bar
   logger.info(c);
});

@Override
public void onBar(BarVO bar) {
   // add bar to the aggregation
   aggregation.add(bar);
}

18.3. Numeric Precision

In General different Securities are traded with different numeric precision (e.g. S&P Futures prices are 2 digits, whereas FX prices are usually 5-6 digits and crypto currencies are up to 8 digits). To accommodate different numeric precisions, AlgoTrader provides the following fields inside the class Security:

- minQty: The minimum tradable quantity
- maxQty: The maximum tradable quantity
• *qtyIncr*: Minimum quantity increase
• *minPrice*: The minimum price
• *maxPrice*: The maximum price
• *priceIncr*: Minimum price increase
• *minNotional*: Minimum order value (quantity x price)

There are also other parameters that describe order minimums and maximums for quantity and price. They are described in section: *Section 23.2.2, “Crypto-Order Constraints”*

### 18.4. Price normalization

Price normalization comes into play when multiple market data and/or trading adapters are in use that use different price multipliers / contract sizes for the same instrument. For example, one adapter might quote prices in dollars with a contract size of 10 where as another one might quote them in cents with a contract size of 1000. The default contract size will be stored in the `security_family` table.

### 18.5. Market Data Gap Checking

Since a continuous data feed of market data is essential for most trading strategies, AlgoTrader contains a feature that automatically warns if no market data has been received for a prolonged period of time. For this purpose, the class `Security` has a property `maxGap`, that defines the maximum number of minutes allowed without any market data updates. This is enforced by the `CHECK_SECURITY_TICK_GAPS` statement which can be turned on by changing the following property inside `conf-core.properties`. Alternatively, the properties can be changed via *Section 2.4, “VM Options”*

```java
# enables security tick gap check
statement.securityTickGap = true
```

Crypto currency exchanges are typically using web sockets to deliver market data. Web socket connections are typically not very stable and it can happen that a web socket connection disconnects or suddenly stops receiving data. For this purpose, AlgoTrader has a feature that automatically reconnects the corresponding adapter if no market data has been received for a prolonged period of time. This is enforced by the `CHECK_ADAPTER_TICK_GAP` statement which can be turned on by changing the following property inside `conf-core.properties`. Alternatively, the properties can be changed via *Section 2.4, “VM Options”*

```java
# enables adapter tick gap check
statement.adapterTickGap = true
```

### 18.6. Generic Events

In addition to MarketDataEvents (i.e. TickVOs and BarVOs) there are general purpose Generic Events that can contain any type of information (e.g. virtual market data, signals, exposure values, etc.). A Generic Event class has to subclass `GenericEventVO`. Subscription to Generic Events is based on the class of the Generic Event. There are two typical use cases for Generic Events:
**External data:** For example feeding Corporate Actions events from external data provider. This use case requires an external data adapter to be implemented. AlgoTrader has an integrated dividend data feed to demonstrate the feature (through the Intrinio data provider). A strategy can subscribe to these events and use them as market signals or inputs. Our Appendix J, Example Strategy “Dividend Capture” shows how this is done.

Example: To subscribe to a Generic Event of type DividendEventVO (which is a subclass of GenericEventVO), the following needs to be done within the strategy:

```java
public void onGenericEvent(final GenericEventVO event) {
    if (event instanceof DividendEventVO) {
        //handle Dividend event
    }
}
```

In distributed mode AlgoTrader must be started with the iNTRDividendGenericEvents Spring profile which would enable the dividend data feed while the strategy must be started with genericEventsService profile, which enables dividend data propagation between server and strategy.

**Strategy to strategy communication:** two (or more) strategies can exchange information between each other. A custom object extending GenericEventVO must be introduced and both strategies must have references to it.

A subscriber strategy can subscribe to particular events in the following way:

```java
getSessionService().subscribeGenericEvents(Collections.singleton(CustomEventVO.class))
```

A publisher strategy can emit custom events like this in distributed mode:

```java
eventDispatcher.broadcast(customEvent, EventRecipient.REMOTE_ONLY);
```

And in embedded mode:
eventDispatcher.broadcast(customEvent, EventRecipient.ALL_LOCAL_LISTENERS)

Alternatively instead of broadcasting, it is possible to send the events directly to the desired strategy as follows:

eventDispatcher.sendEvent(strategy.getName(), customEvent);

If the event is sent directly to the strategy, the target strategy doesn't need to be explicitly subscribed to this event.

All generic events can be persisted in InfluxDB by setting the following property in conf.properties:

```#Defines whether or not to record generic events
dataSource.persistGenericEvents = true```

Alternatively it can be overridden as a VM parameter.

Downloaded (or recorded) generic events can be used in simulations (both InfluxDB and CSV files are supported).

It is also possible to feed Generic Events via CSV Files or InfluxDB in Simulation Mode. The following properties control the data sources:

```#sample CSV dataSource configuration
dataSource.0.subsetName = currentTick
dataSource.0.dataSourceType = CSV
dataSource.0.eventType = TICK
dataSource.0.dataFilesDir = files```

```#sample Influx dataSource configuration
dataSource.1.dataSourceType = DB
dataSource.1.eventType = DIVIDENDS```

The configuration above will feed tick data from CSV and dividend data from InfluxDB. See more sample configurations in conf.properties

The file name of the CSV File has to be according to this schema:

```<className>.<rank>.csv```

- `className` is the fully qualified class name (e.g. `ch.algotrader.event.Signal`)
• *rank* is the sort order for situations where there are multiple *GenericEvent* types for the same time stamp

Suppose you have created your own class extending *GenericEventVO*, say *MyEvent*. In order to make it persistable into InfluxDb, complete the following steps:

• Add an empty public constructor to your class

• Add a *org.influxdb.annotation.Measurement* annotation to the class declaration

• Add at least one field with *org.influxdb.annotation.Column* annotation to the class

```java
@Measurement(name = "MyEvent")
public class MyEvent extends GenericEventVO {

    @Column(name = "Info")
    private String info;

    public MyEvent(){}

    public MyEvent(Instant time, AdapterType adapterType, String id, String info)
    {
        super(time, adapterType, id);
        this.info = info;
    }
}
```

• Configure the DB datasource for your class

```java
dataSource.N.dataSourceType = DB
dataSource.N.eventType = CUSTOM
dataSource.N.fullyQualifiedName = namespace.MyEvent
```

where *N* is the ordinal of the datasource configuration entry, *namespace.MyEvent* is the fully qualified name of your class
Historical Data

AlgoTrader provides Historical Data Interfaces with the following market data providers:

- Native Interfaces
  - Bloomberg
  - Interactive Brokers
  - Quandl
- Crypto REST/WebSocket Interfaces
  - CoinAPI
  - CoinMarketCap

AlgoTrader uses the time series database *InfluxDB* for storage of historical data. InfluxDB is an open source database written in Go specifically designed to handle time series data with high availability and high performance requirements.

In addition the platform also provides a feature for downloading historical data from external market data providers. This historical data can be used for strategy simulations or for any type of analysis.

The Historical Data Manager (*Section 11.3, “Historical Data Manager”*) client provides a convenient UI for managing historical data.

The *HistoricalDataService* provides all relevant functions for historical data:

- **Retrieval**: *e.g.* `getTicksByMaxDate`, `getLastNBarsBySecurityAndBarSize`, etc.
- **Storage**: `storeHistoricalBars`
- **Download**: *e.g.* `downloadHistoricalBars`, `downloadHistoricalTicks`, etc.

To use the Historical Data Service the corresponding Spring profile has to be added via VM argument:

**Bloomberg Historical Data Service:**

```
-Dspring.profiles.active=influxDB,bBHistoricalData
```

**InteractiveBrokers Historical Data Service:**

1 [https://www.influxdata.com/products/influxdb-overview/](https://www.influxdata.com/products/influxdb-overview/)
<table>
<thead>
<tr>
<th>Application</th>
<th>Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quandl Historical Data Service</td>
<td><code>-Dspring.profiles.active=influxDB,qDLHistoricalData</code></td>
</tr>
<tr>
<td>CoinAPI Historical Data Service</td>
<td><code>-Dspring.profiles.active=influxDB,cNPHistoricalData</code></td>
</tr>
<tr>
<td>CoinMarketCap Historical Data Service</td>
<td><code>-Dspring.profiles.active=influxDB,cMCHistoricalData</code></td>
</tr>
<tr>
<td>Noop Historical Data Service</td>
<td><code>-Dspring.profiles.active=influxDB,noopHistoricalData</code></td>
</tr>
</tbody>
</table>

**Note**
The Noop Historical Data Service does not have a connection to an external data source. It can be used during Simulation to access existing historical data from InfluxDB.

**Note**
When running a strategy in distributed mode in case that requires historical data service, a special `historicalData` profile must be specified on the strategy side. In addition an actual historical data profile (e.g. `iBHistoricalData`) must be specified on the server side.

```
-Dspring.profiles.active=live,historicalData
```

### 19.1. InfluxDB

For detailed information on InfluxDB please have a look at the [InfluxDB Documentation](https://docs.influxdata.com/influxdb/).

InfluxDB can be installed locally or using Docker, please see *Chapter 2, Installation and Deployment*.

Data in an InfluxDB instance can be managed via the AlgoTrader Historical Data Manager (*Section 11.3, “Historical Data Manager”*).
InfluxDB provides both a command line client (CLI) as well as a REST based client which is used by various client side libraries. AlgoTrader uses the `influxdb-java`\(^3\) library to communicate with InfluxDB. For all operations that involve the time series database InfluxDB, the following spring profile has to be specified via VM argument:

```
-Dspring.profiles.active=influxDB
```

If InfluxDB is installed locally, the `influx` command should be available via the command line. Executing `influx` will start the CLI and automatically connect to the local InfluxDB instance. The output should look like this:

```
$ influx -precision rfc3339
Connected to http://localhost:8086 version 1.1.x
InfluxDB shell 1.1.x
>
```

Note

- The InfluxDB HTTP API runs on port 8086 by default. Therefore, `influx` will connect to port 8086 and `localhost` by default. If these defaults need to be altered please run `influx --help`.

- The `-precision` argument specifies the format/precision of any returned timestamps. In the example above, `rfc3339` tells InfluxDB to return timestamps in RFC3339 format\(^4\) (YYYY-MM-DDTHH:MM:SS.nnnnnnnnnZ).

The command line is now ready to take input in the form of the Influx Query Language (a.k.a. InfluxQL) statements. To exit the InfluxQL shell, type `exit` and hit return.

Most InfluxQL statements must operate against a specific database. The CLI provides a convenience statement, `USE <db-name>`, which will automatically set the database for all future requests. To use the `algotrader` database please type:

```
> USE algotrader
Using database algotrader
>
```

Now future commands will only be run against the `algotrader` database.

At this point SQL-like queries can be executed against the database. In InfluxDB tables are called measurements. AlgoTrader uses the two measurements `tick` and `bar`. Columns that hold actual data (e.g. `open` or `high`) are called fields, and columns holding static data (e.g. `barSize`) are called tags.

\(^3\) [https://github.com/influxdata/influxdb-java](https://github.com/influxdata/influxdb-java)

\(^4\) [https://www.ietf.org/rfc/rfc3339.txt](https://www.ietf.org/rfc/rfc3339.txt)
As an example the following query shows all current bars in the database:

```
> select * from bar
name: bar
<table>
<thead>
<tr>
<th>time</th>
<th>barSize</th>
<th>close</th>
<th>adapterType</th>
<th>high</th>
<th>low</th>
<th>open</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-01-02T16:48:05Z</td>
<td>MIN_1</td>
<td>116.41</td>
<td>IB</td>
<td>116.42</td>
<td>116.41</td>
<td>104</td>
</tr>
<tr>
<td>2017-01-02T16:49:05Z</td>
<td>MIN_1</td>
<td>116.44</td>
<td>IB</td>
<td>116.44</td>
<td>116.4</td>
<td>104</td>
</tr>
<tr>
<td>2017-01-02T16:50:04Z</td>
<td>MIN_1</td>
<td>116.44</td>
<td>IB</td>
<td>116.44</td>
<td>116.44</td>
<td>104</td>
</tr>
<tr>
<td>2017-01-02T16:59:00Z</td>
<td>MIN_1</td>
<td>116.49</td>
<td>IB</td>
<td>116.51</td>
<td>116.44</td>
<td>104</td>
</tr>
</tbody>
</table>
```

For an in depth description of the query syntax please visit the InfluxDB query language documentation⁵.

To import existing data into InfluxDB please use the following command:

```
> influx -import -path <path-to-file>
```

To import bar data the import file has to be formatted as follows.

```
# DML
# CONTEXT-DATABASE: algotrader

bar,securityId=25,adapterType=IB,barSize=MIN_1
    open=1.30319,high=1.30402,low=1.30319,close=1.30367,vol=0 1324245720000000000
bar,securityId=25,adapterType=IB,barSize=MIN_1
    open=1.30369,high=1.30369,low=1.30351,close=1.30352,vol=0 1324245780000000000
bar,securityId=25,adapterType=IB,barSize=MIN_1
    open=1.30382,high=1.30382,low=1.30382,close=1.30382,vol=0 1324245840000000000
```

To import tick data the import file has to be formatted as follows:

```
# DML
```

⁵ https://docs.influxdata.com/influxdb/v1.1/query_language/
For further information on InfluxDB import please visit the InfluxDB documentation\(^6\)

**Note**

- The last column in the import file represents the time stamp, which needs to be defined in nanoseconds since the 1970-01-01.
- It is also possible to compress import files. In this case the command line switch `-compressed` has to be used when importing files.

To export bar data from InfluxDB into the AlgoTrader CSV file format (see Section 19.8, “Market Data File Format”) please use the following command:

```
> influx -execute "SELECT time as dateTime,open,high,low,close,vol FROM bar" -database "algotrader" -format csv -precision ms > bar.csv
```

To export tick data from InfluxDB please use the following command:

```
> influx -execute "SELECT time as dateTime,last,lastDateTime,volBid,volAsk,bid,ask,vol FROM tick" -database "algotrader" -format csv -precision ms > tick.csv
```

\(^6\) https://docs.influxdata.com/influxdb/v1.2/tools/shell/
226

19.2. Live Data Recording

Using InfluxDB it is possible to store tick-level live data for all subscribed instruments while the system is running. To enable this feature the following properties inside `conf-core.properties` has to be enabled. Alternatively the properties can be changed via `Section 2.4, “VM Options”`:

```
# enables market data persistence
statement.persistMarketData = true
```

In addition recorded tick-level data can be aggregated into bar-data on the fly by using the following properties inside `conf-core.properties`. Alternatively the properties can be changed via `Section 2.4, “VM Options”`:

```
# enables market data persistence
statement.aggregateBars = true

# the bar size used for tick-to-bar aggregation and end-of-day historical bar download
historicalData.barSize = MIN_1
```

In case a certain instrument provides trade information (e.g. Equities and Crypto Currencies) the last traded price is used to calculate the Bar values. In case no trading information is available (e.g. Forex and Indices) the midpoint price (average of bid and ask) is used. The bar aggregation feature will also create bars for time periods when no market data arrives, in this case open, high, low and close will be equal to the previous bars close price.

19.3. Historical Data Download

The `storeHistoricalBars` method of the Historical Data Service saves historical bars directly into InfluxDB. If the parameter `replace` is set to false the method `storeHistoricalBars` will save newly retrieved Bars after the last Bar currently in the database. Bars before the current last Bar will not be touched. If the parameter

Note

The InfluxDB export adds an extra column named "name" as the first column. In order to use the exported `.csv` file for simulations one has to remove the first column of the file. The following Linux command can be used to accomplish this:

```
cut --complement -f 1 -d, tick.csv > tick-new.csv
```
replace is set to true the method storeHistoricalBars however will replace all current Bars in the database within the specified time period

Download and storage of historical data can be invoked via the HistoricalDataStarter.

```
HistoricalDataStarter replaceBars startDate endDate marketDataEvent-type barSize securityId(s)
```

For example:

```
HistoricalDataStarter true 2016-01-01 2016-12-31 TRADES DAY_1 10 11 12
```

AlgoTrader also provides features to download missing historical data for all subscribed instruments either on startup or at a specific time of the day. For these functions the following properties are available inside conf-core.properties where they can be changed. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```
# enables end-of-day historical bar download
statement.downloadHistoricalDataEOD = true

# the bar size used for tick-to-bar aggregation and end-of-day historical bar download
historicalData.barSize = MIN_1

# the market data event type used by the end-of-day historical bar download
historicalData.marketDataEvent-type = MIDPOINT

# Hour-of-Day when the end-of-day historical bar download takes place
historicalData.downloadHour = 2

# enables historical bar download on startup
historicalData.downloadHistoricalDataOnStartup = true
```

**Note**

Available market data event types are:

- TRADES
- MIDPOINT
- BID
- ASK
- BID_ASK
- BEST_BID
Depending on whether Interactive Brokers, Bloomberg or Quandl is used for the historical data download the corresponding *marketData* profile has to be specified via VM argument.

Interactive Brokers:

- `-Dspring.profiles.active=influxDB,iBHistoricalData`

Bloomberg:

- `-Dspring.profiles.active=influxDB,bBHistoricalData`

Quandl:

- `-Dspring.profiles.active=influxDB,qdlHistoricalData`

19.4. Interactive Brokers Historical Data Download

The Historical Data Download incorporates *historical data limitations*⁷ in place by Interactive Brokers.

With IB API the following conditions can lead to pacing violations:

- Making six or more historical data requests for the same Contract, Exchange and Tick Type within two seconds.
- Making more than 60 historical data requests in any ten-minute period.

The AlgoTrader Historical Data Download can optionally avoid potential pacing violation by separating subsequent download requests by 10 seconds. This feature can be enabled via the following property inside `conf-ib.properties` has to be updated. Alternatively the properties can be changed via *Section 2.4, “VM Options”*

```java
# ensures 10 seconds delay between historic data requests
ib.pacingViolationGuard = true
```

The Historical Data Download also takes the *Valid Duration and Bar Size Settings*⁸ for Historical Data Requests into account and splits large requests into subsequent smaller requests.

---


19.5. Quandl Historical Data Download

Quandl\(^9\) is a public service that provides a wide range of financial, economic and alternative data. AlgoTrader allows downloading historical data from Quandl. For more information please visit section Section 23.22, “Quandl”.

19.6. CoinAPI Historical Data Download

CoinAPI\(^10\) is a platform which provides fast, reliable unified data APIs to cryptocurrency markets. AlgoTrader allows downloading historical data from CoinAPI. For more information please visit section Section 23.37, “CoinAPI”.

19.7. CoinMarketCap Historical Data Download

CoinMarketCap\(^11\) is a provider for cryptocurrency data. AlgoTrader allows downloading historical data from CoinMarketCap. For more information please visit section Section 23.39, “CoinMarketCap”.

19.8. Market Data File Format

When using CSV files for the back test all data files are placed inside the following directory structure:

```
/<baseDir>/<eventType>/<dataSet>/<filename>.csv
```

- baseDir is the parent directory where all market data files are stored. This is either the files/ directory under the project algotrader-core or an arbitrary directory defined via the following property inside conf.properties has to be updated. Alternatively the properties can be changed via Section 2.4, “VM Options”

  ```
  # alternate dataSetLocation (default is <working-dir>/files/ i.e. usually algotrader/core/files/ )
  dataSource.0.dataSetLocation = files
  ```

- eventType is either tickData, barData or customData (see Section 18.6, “Generic Events”).

- dataSet is the name of the dataset used for the simulation run. This can be defined via the following property inside conf.properties has to be updated. Alternatively the properties can be changed via Section 2.4, “VM Options”

  ```
  # name of dataSet to be used for simulations and market data persistence
  dataSource.0.dataSet = current
  ```

- filename can be either of the following values followed by .csv

---

\(^9\) https://www.quandl.com/
\(^10\) https://www.coinapi.io/
\(^11\) https://coinmarketcap.com/
An alternative approach is to feed market data for multiple securities using one file. E.g. it is possible to feed market data for futures using market data from the corresponding generic future. In this approach an additional column `security` has to be added to the market data file which will be used to identify the actual Security.

The first line within the file is the header row.

The file name for Section 18.6, “Generic Events” follows a different logic.

### 19.8.1. Tick Data Files

The Format of the Tick Data Files is based on a standard CSV Structure:

- `dateTime`
- `last`
- `lastDateTime`
- `volBid`
- `volAsk`
- `bid`
- `ask`
- `vol`
- `security` (optional)

`dateTime` and `lastDateTime` values are expected to be in the `yyyy-MM-dd HH:mm:ss` format and to represent local time. Alternatively one can also use long values that represent Java milliseconds since 1970.

Example:

**Table 19.1. Tick Data Format**

<table>
<thead>
<tr>
<th><code>dateTime</code></th>
<th><code>last</code></th>
<th><code>lastDateTime</code></th>
<th><code>volBid</code></th>
<th><code>volAsk</code></th>
<th><code>bid</code></th>
<th><code>ask</code></th>
<th><code>vol</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-01-01 14:00:01</td>
<td>188</td>
<td>2016-01-01 14:00:01</td>
<td>47</td>
<td>52</td>
<td>178.1</td>
<td>183.2</td>
<td>20</td>
</tr>
</tbody>
</table>
The Format of the Bar Data Files is based on a standard CSV Structure:

- dateTime
- open
- high
- low
- close
- vol
- vwap (optional)
- security (optional)

datetime values are expected to be in the yyyy-MM-dd HH:mm:ss format and to represent local time. Alternatively one can also use long values that represent Java milliseconds since 1970.

Example:

**Table 19.2. Bar Data Format**

<table>
<thead>
<tr>
<th>dateTime</th>
<th>open</th>
<th>high</th>
<th>low</th>
<th>close</th>
<th>vol</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-01-01 14:00:00</td>
<td>1.29366</td>
<td>1.29369</td>
<td>1.29360</td>
<td>1.29369</td>
<td>2000</td>
</tr>
<tr>
<td>2016-01-01 14:01:00</td>
<td>1.29367</td>
<td>1.29389</td>
<td>1.29367</td>
<td>1.29378</td>
<td>2500</td>
</tr>
</tbody>
</table>
Chapter 20.

Reference Data


Reference Data can either be configured in the database directly through the corresponding tables, one can use the ReferenceDataService and corresponding ReferenceDataStarter or you can use the Reference Data Manager UI, see Section 11.2, “Reference Data Manager”.

Depending on the Reference Data Adapter in use the following download options are available for download:

- All Futures of all Security Families (of family type Future)
- All Futures of a particular Security Family
- All Options of all Security Families (of family type Option)
- All Options of a particular Security Family
- All Stocks of a particular Security Family
- All items available through the particular Reference Data Adapter

For further details please see the JavaDoc of the ReferenceDataStarter class.

Example: To download missing Futures of a specified Security Families use the following command

```
ReferenceDataStarter futures securityFamilyId1,securityFamilyId2 ...
```

It is recommended to run this Service in the interval of Option / Future Expirations to make sure that the entire chain is available to strategies.

Depending on the Reference Data Adapter in use the corresponding referenceData profile has to be specified via VM argument.

Bloomberg:

```
-Dspring.profiles.active=<dataSource>,bBReferenceData
```

B2C2:

```
-Dspring.profiles.active=<dataSource>,b2C2ReferenceData
```

Binance:

```
-Dspring.profiles.active=<dataSource>,bNCRferenceData
```
Bitfinex:
-Dspring.profiles.active=<dataSource>,bFXReferenceData

Bitflyer:
-Dspring.profiles.active=<dataSource>,bFLReferenceData

BitHumb Pro:
-Dspring.profiles.active=<dataSource>,bHBReferenceData

BitMEX:
-Dspring.profiles.active=<dataSource>,bMXReferenceData

Bitstamp:
-Dspring.profiles.active=<dataSource>,bTSReferenceData

Bittrex:
-Dspring.profiles.active=<dataSource>,bTXReferenceData

CoinAPI:
-Dspring.profiles.active=<dataSource>,cNPReferenceData

Coinbase Pro:
-Dspring.profiles.active=<dataSource>,cNBReferenceData

CoinMarketCap:
-Dspring.profiles.active=<dataSource>,cMCReferenceData

CryptoFinance:
-Dspring.profiles.active=<dataSource>,cFReferenceData

Deribit:
-Dspring.profiles.active=<dataSource>,dRBReferenceData
<table>
<thead>
<tr>
<th>Platform</th>
<th>Profiles Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enigma</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,eGMReferenceData</td>
</tr>
<tr>
<td>eToroX</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,eTXReferenceData</td>
</tr>
<tr>
<td>Huobi Spot</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,hBIReferenceData</td>
</tr>
<tr>
<td>InteractiveBrokers</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,iBReferenceData</td>
</tr>
<tr>
<td>Kraken Spot</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,kKSReferenceData</td>
</tr>
<tr>
<td>OKEx/OKCoin</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,oKXReferenceData</td>
</tr>
<tr>
<td>Tilde</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,tLDReferenceData</td>
</tr>
<tr>
<td>CryptoFinance</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,cFReferenceData</td>
</tr>
<tr>
<td>Enigma</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,eGMReferenceData</td>
</tr>
<tr>
<td>LMAX Digital</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,lMAXDReferenceData</td>
</tr>
<tr>
<td>OKEx</td>
<td>-Dspring.profiles.active=&lt;dataSource&gt;,oKEXReferenceData</td>
</tr>
</tbody>
</table>
OKCoin:

-Dspring.profiles.active=<dataSource>,oKCoinReferenceData

Trading Technologies:

-Dspring.profiles.active=<dataSource>,tTReferenceData

Note

When running a strategy in distributed mode that requires reference data service, a special referenceData profile must be specified on the strategy side. In addition an actual historical data profile (e.g. iBReferenceData) must be specified on the server side.

-Dspring.profiles.active=live,referenceData
Account Data

AlgoTrader provides Account Data Service for the following adapters:

- Interactive Brokers via Native API
- Binance via Binance API
- Bitfinex via Bitfinex API
- Bitflyer via Bitflyer API
- BitHumb via BitHumb API
- BitMEX via BitMEX API
- Bittrex via Bittrex API
- Bitstamp via Bitstamp API
- Coinbase Pro via Coinbase API
- Deribit via Deribit API
- eToroX via eToroX API
- Gemini via Gemini Rest API
- Huobi Spot via Huobi Spot API
- Kraken Spot via Kraken Websockets API and Kraken Rest API
- OKEx via OKEx API
- OKCoin via OKCoin API

---

1 https://github.com/binance-exchange/binance-java-api
2 https://docs.bitfinex.com/docs
3 https://bitflyer.com/api
4 https://github.com/bithumb-pro/bithumb.pro-official-api-docs
5 https://www.bitmex.com/app/apiOverview
6 https://bittrex.github.io/api/v1-1
7 https://www.bitstamp.net/api/
8 https://docs.pro.coinbase.com/
9 https://docs.deribit.com/v2/
10 https://etorox.github.io/docs/#/README
11 https://docs.gemini.com/rest-api/
12 https://huobiapi.github.io/docs/spot/v1/en/#introduction
13 https://docs.kraken.com/websockets/
14 https://www.kraken.com/features/api
16 https://www.okcoin.com/docs/en/
Depending on the Account Data Adapter in use the corresponding account profile has to be specified via VM argument.

InteractiveBrokers:

-Dspring.profiles.active=iBAccount

Binance:

-Dspring.profiles.active=bNCAccount

Bitfinex:

-Dspring.profiles.active=bFXAccount

Bitflyer:

-Dspring.profiles.active=bFLAccount

BitHumb Pro:

-Dspring.profiles.active=bHBAccount

BitMEX:

-Dspring.profiles.active=bMXAccount

Bitstamp:

-Dspring.profiles.active=bTSAccount

Bittrex:

-Dspring.profiles.active=bTXAccount

Coinbase Pro:

-Dspring.profiles.active=cNBAccount

Deribit:

-Dspring.profiles.active=dRBAccount

Gemini:
Huobi Spot:

-Dspring.profiles.active=hBIAccount

Kraken Spot:

-Dspring.profiles.active=kKSAccount

OKEx:

-Dspring.profiles.active=oKEXAccount

OKCoin:

-Dspring.profiles.active=oKCoinAccount

Note

When running a strategy in distributed mode in case that requires account data service, a special account profile must be specified on the strategy side. In addition an actual historical data profile (e.g. iBAccount) must be specified on the server side.

-Dspring.profiles.active=live,account

21.1. Account balances

The AccountService interface defines a method for retrieving account balances for the specified account ID. A list of NamedCurrencyAmountVO items is returned. The retrieveAccountBalances method can be called from the strategy like this:

```
List<NamedCurrencyAmountVO> balances = getAccountService().retrieveAccountBalances(accountId);
```

21.2. Withdrawal

The AccountService interface defines a method to initiate a crypto withdrawal for crypto exchanges. The withdraw method can be called from the strategy like this:

```
WithdrawStatusVO status = getAccountService().withdraw(accountId, currency, amount, withdrawContext);
```
The `withdrawContext` parameter contains additional information that might be required by certain exchanges (e.g., `address` and/or `paymentId`). The method returns a `WithdrawStatusVO` which contains information like `message` and `externalId`.

For withdrawal operations exchanges typically require additional security constraints, which might include:

- API key with explicit permission to withdraw
- Two-factor authentication (2FA)
- IP address whitelisting
- Deposit address whitelisting
- Email confirmation

Please consult the corresponding exchange's documentation about the details on how to enable withdrawals.

**Note**

Typically each adapter has a configurable list of supported currencies, for example for Bitfinex this list is configured in `conf-bfx.properties` file. This is adapter specific, but often a withdrawal operation is only allowed if the currency is explicitly listed (please check the corresponding adapter section).

### 21.3. Deposit address

The `AccountService` provides a method for getting the deposit address. The method can be called from a strategy like this:

```java
String depositAddress = getAccountService().getDepositAddress(DepositAddressVo depositParams);
```

This returns the deposit address for an exchange, currency and wallet type (wallet type is required for Bitfinex only). It returns the value as a String. Please check the exchange documentation on how to create deposit addresses. The `getDepositAddress` method sends the request to an enabled exchange (see Section 26.2, "Spring Profiles"). If the requested exchange profile is not present, an exception is thrown. Some exchanges support different currencies (please check the corresponding adapter section).

### 21.4. Account Events

Using the `AccountService` it is also possible to subscribe for account events. After successful subscription the strategy will be notified each time the account balances are changed.

In order to subscribe the strategy needs to call
getAccountService().subscribeAccountEvent(accountId);

To process events the strategy must implement a listener method:

```java
void onAccountEvent(AccountEventVO event)
```

The `AccountEventVO` has a method `getAccountBalances` which returns the received balances.

```java
List<NamedCurrencyAmountVO> getAccountBalances();
```

Unlike the `retrieveAccountBalances` which always return all balances, the account update event may contain only the balance which has changed.

**Note**

Subscription for account events is supported only by selected exchanges (please check the corresponding adapter section).
AlgoTrader API

The following sections describe how the AlgoTrader system can be accessed by external systems.

22.1. JSON data binding

AlgoTrader uses a consistent message format based on JSON for both the WebSocket/STOMP API and the REST API.

Using the JSON messages in combination with the AlgoTrader REST API and WebSocket/STOMP API any popular development language can be used to build trading strategies making use of the AlgoTrader platform, for example:

- C#
- C++
- Python
- R
- MatLab
- JavaScript / NodeJS

22.2. REST API

Access to most of the functionality and data provided by AlgoTrader is available via a REST based API.

The full documentation of the AlgoTrader REST API is available here.¹

HTTP GET endpoints can easily be queried via the Browser. To retrieve a JSON formatted list of all accounts open to the following URL in the Browser:

http://localhost:9090/rest/account

In addition one can use Curl², a popular utility for execution of HTTP requests. The following example shows how to retrieve a JSON formatted list of all accounts by executing HTTP GET request:

```
$ curl -X GET http://localhost:9090/rest/account -i
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(9.3.6.v20151106)
```

¹ http://doc.algotrader.com/apidocs/index.html
² https://curl.haxx.se/
Similarly one can request AlgoTrader to subscribe to market data for security with id 11 by executing HTTP PUT request:

```bash
$ curl -X PUT -H "content-type: application/json" 
   http://localhost:9090/rest/subscription/marketdata/subscribe 
   -d "{"strategyName":"SERVER","securityId":11,"subscribe":true}" -i 
HTTP/1.1 200 OK 
Content-Length: 0 
Server: Jetty(9.3.6.v20151106)
```

22.3. WebSocket/STOMP API

AlgoTrader employs STOMP messaging protocol over WebSockets transport to implement multi-topic, multi-client message delivery based on the Publish-Subscribe pattern. AlgoTrader acts as a message producer that generates messages representing various system or trading related events and publishes them to predefined topics. Browsers running the AlgoTrader UI (and potentially any external application supporting STOMP over WebSockets) act as message consumers that subscribe to message topics of interest such as market data, orders, order status updates, position changes, executed transactions and so on. Consumers express their interest in a particular type of event by subscribing to message topics. Consumers should no longer need to filter out unwanted messages. They are expected to subscribe only to a subset of messages they are interested in.

For further details on the STOMP protocol please visit the [STOMP website](https://stomp.github.io/).

AlgoTrader publishes events to multiple predefined event topics. Note: for each topic AlgoTrader creates additional topic related to particular strategy, order, instrument.

For example: all ticks for security with ID 53 are published to topic tick.53, for instrument ID 46: tick.46

To receive all the ticks one need to subscribe for topic tick.>

You can see all available topics and description of its content via Rest request, e.g. if you have AlgoTrader running on your local machine: [http://localhost:9090/stomp-info.html](http://localhost:9090/stomp-info.html)

Some topics are listed below. Please note: content can be either of class specified below or a subclass. See more details in [apidocs](https://doc.algotrader.com/apidocs/index.html#syntax_json)
### Table 22.1. Event topics

<table>
<thead>
<tr>
<th>Topic format</th>
<th>Type of events</th>
</tr>
</thead>
<tbody>
<tr>
<td>/topic/generic-event.&lt;StrategyName&gt;.&lt;EventId&gt;</td>
<td>GenericEventVO</td>
</tr>
<tr>
<td>/topic/ib-generic-event.&lt;StrategyName&gt;.&lt;EventId&gt;</td>
<td>IBGenericTickVO</td>
</tr>
<tr>
<td>/topic/order-request-result.&lt;StrategyName&gt;.&lt;OrderId&gt;</td>
<td>OrderRequestStatusEventVO</td>
</tr>
<tr>
<td>/topic/tick.&lt;security_id&gt;</td>
<td>TickVO</td>
</tr>
<tr>
<td>/topic/market-data-subscription.&lt;strategy_name&gt;.&lt;security_id&gt;.&lt;adapter_type&gt;</td>
<td>MarketDataSubscriptionVO</td>
</tr>
<tr>
<td>/topic/log-event.&lt;LogLevel&gt;</td>
<td>LogEventVO</td>
</tr>
<tr>
<td>/topic/quote.&lt;QuoteId&gt;</td>
<td>SingleQuoteEventVO</td>
</tr>
<tr>
<td>/topic/transaction.&lt;strategy_name&gt;.&lt;uuid&gt;</td>
<td>TransactionVO</td>
</tr>
<tr>
<td>/topic/algo-order-state.&lt;StrategyName&gt;.&lt;AlgoOrderIntId&gt;</td>
<td>AlgoOrderStateVO</td>
</tr>
<tr>
<td>/topic/position.&lt;strategy_name&gt;.&lt;id&gt;</td>
<td>PositionVO</td>
</tr>
<tr>
<td>/topic/order.&lt;strategy_name&gt;.&lt;order_int_id&gt;</td>
<td>OrderVO</td>
</tr>
<tr>
<td>/topic/order-status.&lt;strategy_name&gt;.&lt;order_int_id&gt;</td>
<td>OrderStatusVO</td>
</tr>
<tr>
<td>/topic/cash-balance.&lt;strategy_name&gt;.&lt;order_int_id&gt;</td>
<td>CashBalanceVO</td>
</tr>
<tr>
<td>/topic/order-book.&lt;SecurityId&gt;</td>
<td>OrderBookVO</td>
</tr>
<tr>
<td>/topic/bar.&lt;SecurityId&gt;</td>
<td>BarVO</td>
</tr>
<tr>
<td>/topic/account-event.&lt;account_id&gt;</td>
<td>AccountEventVO</td>
</tr>
</tbody>
</table>

Example of the content for some topics are provided below:

**TickVO**

```json
{
  "dateTime": 1585146085447,
  "adapterType": "CNB",
  "securityId": 854,
  "last": 6607.47,
  "lastDateTime": 1585146083014,
  "bid": 6606.2,
  "ask": 6607.47,
  "volBid": 0.0018634,
  "volAsk": 0.91054995,
  "vol": 36881.57425883,
  "objectType": "TickVO",
  "currentValue": 6606.84
}
```
SingleQuoteEventVO

```json
{
    "id": 574,
    "intId": "546",
    "extId": "Asde",
    "side": "BUY",
    "bidPx": 86.99,
    "bidSize": 1200000,
    "askPx": 87.22,
    "askSize": 1800000,
    "dateTime": 1585141115408,
    "validUntil": 1585145145648,
    "quoteRequestId": 2136068173,
    "securityId": 184,
    "accountId": 19544,
    "objectType": "SingleQuoteEventVO"
}
```

OrderVO (example of subclass MarketOrderVO)

```json
{
    "id": 520434899,
    "intId": "43563564",
    "extId": "rwAXs",
    "parentIntId": "56434",
    "dateTime": 1585145145648,
    "side": "SELL_SHORT",
    "quantity": 6859,
    "tif": "FOK",
    "tifDateTime": 1585145145648,
    "exchangeOrder": false,
    "exchangeId": 1651,
    "parentOrderID": 433384635,
    "securityId": 1132,
    "accountId": 12971,
    "strategyId": 21,
    "lastStatus": "EXECUTED",
    "objectType": "MarketOrder"
}
```
OrderStatusVO

{
  "id": 627598574,
  "dateDateTime": 1585145145648,
  "extDateTime": 1585145145648,
  "status": "EXECUTED",
  "filledQuantity": 1298227,
  "remainingQuantity": 0.0,
  "lastQuantity": 100000,
  "avgPrice": 363.62,
  "lastPrice": 363.02,
  "intId": "1254",
  "extId": "aret",
  "sequenceNumber": 134,
  "reason": "",
  "orderId": 2063904335,
  "objectType": "OrderStatus"
}

OrderBookVO

{
  "dateDateTime": 1585146085447,
  "adapterType": "CNB",
  "securityId": 854,
  "id": 0,
  "bids": {
    "6606.20": {
      "price": 6606.2,
      "amount": 0.0018634,
      "count": 0
    },
    "6605.58": {
      "price": 6605.58,
      "amount": 0.25,
      "count": 0
    },
    "6494.51": {
      "price": 6494.51,
      "amount": 0.001,
      "count": 0
    }
  }
}
Topics are organized by name spaces. A consumer wishing to receive market data for security with id 12 only can subscribe to the following topic:

/topic/tick.12

A consumer wishing to receive market data all securities can subscribe to the following wild card topic.

/topic/tick.*

Strategy specific events are organized by strategy name. A consumer wishing to receive order status updates for the order with internal id 10 issued by strategy MY_STRATEGY can subscribe to the following topic:

/topic/order-status.MY_STRATEGY.10

A consumer wishing to receive order status updates for all orders issued by strategy MY_STRATEGY can subscribe to the following wild card topic
The * wild card selects all elements within the same namespace

A consumer wishing to receive order status updates for all orders of all strategies can subscribe to the following wild card topic

```
/topic/order-status.MY_STRATEGY.*
```

The > wild card selects all topics within the same namespace and their sub-namespaces.

In order to ensure optimal performance of HTML5 clients AlgoTrader can throttle market data event delivered by the WebSockets transport. The embedded message broker by default attempts to ensure that the total rate of events per connection does not exceed 50 per second. At the same time instruments with infrequent market data updates are not throttled if their total event rate is below 0.1 per second (less than one event every 10 seconds).

Throttling rates can be adjusted by changing the following configuration parameters:

```
activeMQ.maxRatePerConnection = 50
activeMQ.minRatePerConsumer = 0.1
```

Websocket(STOMP) port is configured here (with default value)

```
activeMQ.stomp.port = 61617
```

In JavaScript STOMP messages can be consumed like this:

```
<html>
  <head>
    <script src="https://unpkg.com/@stomp/stompjs@4.0.6/lib/stomp.min.js"></script>
    <script type="text/javascript">
      var ws = new WebSocket("ws://localhost:61614", "stomp");
      var stompClient = Stomp.over(ws);
      stompClient.connect({}, function(frame) {
        stompClient.subscribe('/topic/tick.*', function(message){
          console.log(JSON.parse(message.body));
        });
      });
    </script>
  </head>
</html>
```

For further details please visit the *STOMP JavaScript documentation*.

5 http://jmesnil.net/stomp-websocket/doc/
AlgoTrader supports STOMP over normal TCP connections. It designed initially for Python clients.

Sample subscription consumer code in Python:

```python
import time
import sys

import stomp

class MyListener(stomp.ConnectionListener):
    def on_error(self, headers, message):
        print('received an error "%s" % message)
    def on_message(self, headers, message):
        print('received a message "%s" % message)

hosts = [('localhost', 61618)]

conn = stomp.Connection(host_and_ports=hosts)
conn.set_ssl(for_hosts=hosts)
conn.set_listener('', MyListener())
conn.start()
print('started')
conn.connect()
conn.subscribe(destination='/topic/log-event.error', id='sub-0', ack='auto')
conn.subscribe(destination='/topic/tick.849', id='10', ack='auto')

time.sleep(10000)
conn.disconnect()
```

To consume messages from java program the following code snippet provides details:

```java
import org.apache.activemq.transport.stomp.Stomp;
import org.apache.activemq.transport.stomp.StompConnection;
import org.apache.activemq.transport.stomp.StompFrame;
import org.json.JSONObject;

public class StompConsumer {
    public static void main(String[] args) throws Exception {
        StompConnection connection = new StompConnection();
        connection.open("localhost", 61617);
        connection.connect(null, null);
        StompFrame connect;
        connection.subscribe("/topic/tick.1239", Stomp.Headers.Subscribe.AckModeValues.AUTO);

        while (true) {
```
connect = connection.receive();

String bodyStr = connect.getBody();
if (bodyStr.isEmpty()) {
    continue;
}

JSONObject jsonObject = new JSONObject(bodyStr);
if (jsonObject.has("objectType")) {
    String objectType = jsonObject.getString("objectType");
    if ("Tick".equals(objectType)) {
        System.out.println("Tick message:
" + connect.getHeaders());
        System.out.println("Content:
" + new String(connect.getContent()));
        return;
    }
}
}

Example output:

Tick message:
{content-length=227, expires=0, destination=/topic/tick.1239, priority=4, message-id=ID:DESKTOP-TE1MV6G-59998-1584355272592-4:1:10:1:7917, persistent=true, timestamp=1584356107716}
Content:
{"dateTime":1584356107715,"adapterType":"CNB","securityId":1239,"last":139.38,"lastDateTime":1584356106994,"bid":139.26,"ask":139.8,"volBid":2.0375,"volAsk":0.3245,"vol":13729.06555054,"objectType":"Tick","currentValue":139.53}

22.4. Inbound FIX API

AlgoTrader provides Inbound FIX protocol based API which allows submission and cancellation of simple orders (Market, Limit, Stop, Stop Limit) and algo orders (TWAP, VWAP and other types of algo order including smart order routing (SOR)). Those are then forwarded via the configured trading adapter to the relevant broker / exchange. You can also configure AlgoTrader to connect to multiple exchanges and use Inbound FIX API to submit orders to any of them or use SOR to let AT choose how to route the order.

Order status changes from the broker/exchange are then forwarded back to the external FIX client. This mechanism allows using the industry standard FIX API for order management via all trading adapters AlgoTrader provides. Thus also for exchanges or brokers that do not natively support FIX.

The following is an example of the QuickFixJ config (fix.cfg) that you can use to make the Inbound FIX API available.
You can test AlgoTrader Inbound FIX API using test client (code path: tests/integration/src/main/java/quickfix - based on code of Banzai FIX GUI client).

The following is an example banzai.cfg config for connecting to locally running AlgoTrader:

```
[default]
FileStorePath=target/data/banzai
ConnectionType=initiator
SenderCompID=BANZAI
TargetCompID=AT
SocketConnectHost=localhost
SocketConnectPort=9880
UseDataDictionary=Y
DataDictionary=inboundfix/FIX44.xml
StartTime=00:00:00
EndTime=00:00:00
HeartBtInt=30
ReconnectInterval=5
```

This is an example runner configuration for starting AlgoTrader with Inbound FIX API enabled and connected to Interactive Brokers:

---

6 https://github.com/quickfix-j/quickfixj/tree/master/quickfixj-examples/banzai
Main Class: ch.algotrader.starter.ServerStarter
VM Options: -Dspring.profiles.active=live,pooledDataSource,embeddedBroker,inboundFix,iBNative,iBMarketData,html5

Note that you can open the HTML5 UI (http://localhost:9090) and the orders you place via Inbound FIX API will be visible there. QuickFixJ logs with FIX messages between Banzai and AlgoTrader can be found in \log \FIX.4.4-AT-BANZAI.messages.log file. The raw FIX messages can be translated by e.g. FIX decoder\footnote{https://drewnoakes.com/fix-decoder/}.

The protocol version provided by the AlgoTrader Inbound FIX API is FIX 4.4. We use standard messages with some custom tags and values which are provided below:

- ExchangeOrder(20003) - true if it is an exchange order, false otherwise
- NoRoutingCandidates(20020) - repeating group tag. This group defines routing candidates for smart order routing.
  
  Note: either SOR must be defined or triplet: {Account, SecurityId, Exchange}, exchange is optional
  
  CandidateSecurityId(20021) - routing candidate Security ID, type int
  
  CandidateAccountId(20022) - routing candidate Account ID, type int
  
  CandidateExchangeId(20023) or CandidateExchangeName(20024) - routing candidate exchange ID or name respectively, type int or String

The following messages are supported:

- Logon (A)
- Logout (5)
- Test Request (1)
- Heartbeat (0)
- Resend Request (2)
- Execution Report (8)
- New Order Single (D)
- Business Message Reject (j)
- Order Cancel Request (F)
- Order Cancel Reject (9)
22.4.1. Logon message

Used to connect to AlgoTrader. Sent from external client to AlgoTrader.

Table 22.2. Logon

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>A</td>
</tr>
<tr>
<td>98</td>
<td>EncryptMethod</td>
<td>Y</td>
<td>Char</td>
<td>Should be '0' (NONE_OTHER) unless Inbound FIX API configured differently.</td>
</tr>
<tr>
<td>141</td>
<td>ResetSeqNumFlag</td>
<td>Y</td>
<td>Char</td>
<td>'N' - Default. Sequence numbers should not be reset. 'Y' - Sequence numbers should be reset.</td>
</tr>
<tr>
<td>108</td>
<td>HeartBtInt</td>
<td>N</td>
<td>Char</td>
<td>Interval in seconds. Default: 30.</td>
</tr>
</tbody>
</table>

22.4.2. Logout message

Used to disconnect from AlgoTrader. Sent from external client to AlgoTrader.

Table 22.3. Logout

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>5</td>
</tr>
<tr>
<td>58</td>
<td>Text</td>
<td>N</td>
<td>String</td>
<td>Some arbitrary text value to indicate the intention to Logout.</td>
</tr>
</tbody>
</table>

22.4.3. Test Request message

The TestRequest message forces a Heartbeat from the opposing application. Sent from external client to AlgoTrader or vice versa.

Table 22.4. Test Request

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>112</td>
<td>TestReqID</td>
<td>N</td>
<td>String</td>
<td>The value in this field to be returned with the Heartbeat message (MsgType = 0).</td>
</tr>
</tbody>
</table>

22.4.4. Heartbeat message

Used to verify the connection with AlgoTrader. Sent from external client to AlgoTrader or vice versa.
Table 22.5. Heartbeat

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>0</td>
</tr>
<tr>
<td>112</td>
<td>TestReqID</td>
<td>N</td>
<td>String</td>
<td>Identifier included in Test Request message to be returned in resulting Heartbeat.</td>
</tr>
</tbody>
</table>

22.4.5. Resend Request message

The Resend Request message is sent by the client to AlgoTrader to request the retransmission of messages.

Table 22.6. Resend Request

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>BeginSeqNo</td>
<td>Y</td>
<td>String</td>
<td>Message number of the first message in range to be resent.</td>
</tr>
<tr>
<td>16</td>
<td>EndSeqNo</td>
<td>Y</td>
<td>String</td>
<td>Message sequence number of last message in range to be resent.</td>
</tr>
</tbody>
</table>

22.4.6. New Order Single message

Used to submit a new order to AlgoTrader. Sent from external client to AlgoTrader.

Table 22.7. New Order Single

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Standard Msg Type</td>
<td>Y</td>
<td>Char</td>
<td>D</td>
</tr>
<tr>
<td>11</td>
<td>ClOrdID</td>
<td>Y</td>
<td>String</td>
<td>Client order ID assigned by the client</td>
</tr>
<tr>
<td>54</td>
<td>Side</td>
<td>Y</td>
<td>Char</td>
<td>1 or 2 for Buy/Sell</td>
</tr>
<tr>
<td>60</td>
<td>TransactTime</td>
<td>Y</td>
<td>UTCTimestamp</td>
<td>UTC time when the message was sent</td>
</tr>
<tr>
<td>40</td>
<td>OrdType</td>
<td>C</td>
<td>Char</td>
<td>required for non algo orders - Market(1), Limit(2), Stop(3), StopLimit(4), must not be specified for Algo order</td>
</tr>
<tr>
<td>38</td>
<td>OrderQty</td>
<td>Y</td>
<td>Qty</td>
<td>Total order quantity</td>
</tr>
<tr>
<td>1</td>
<td>Account</td>
<td>C</td>
<td>String</td>
<td>AlgoTrader account ID, required for non SOR</td>
</tr>
<tr>
<td>48</td>
<td>SecurityID</td>
<td>C</td>
<td>String</td>
<td>AlgoTrader security ID, required for non SOR</td>
</tr>
<tr>
<td>100</td>
<td>ExDestination</td>
<td>N</td>
<td>String</td>
<td>AlgoTrader exchange ID</td>
</tr>
<tr>
<td>Tag #</td>
<td>Field Name</td>
<td>Required</td>
<td>Data type</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------</td>
<td>----------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>207</td>
<td>SecurityExchange</td>
<td></td>
<td>String</td>
<td>AlgoTrader exchange name. Note: if non of fields are set (ExDestination, SecurityExchange), then default exchange will be used</td>
</tr>
<tr>
<td>59</td>
<td>TimeInForce</td>
<td></td>
<td>Char</td>
<td>How long an order remains active. Supported: 0, 1, 2, 3, 4, 6, 7 (depending support by the exchange)</td>
</tr>
<tr>
<td>432</td>
<td>ExpireDate</td>
<td></td>
<td>LocalDate</td>
<td>Required for TIF Good Till Date</td>
</tr>
<tr>
<td>126</td>
<td>ExpireTime</td>
<td></td>
<td>LocalTime</td>
<td>Required for TIF Good Till Date</td>
</tr>
<tr>
<td>44</td>
<td>Price</td>
<td></td>
<td>Price</td>
<td>Required for Limit, StopLimit orders</td>
</tr>
<tr>
<td>7003</td>
<td>ExchangeOrderN</td>
<td></td>
<td>Boolean</td>
<td>'Y' means exchange order and 'N' margin order on certain exchanges.</td>
</tr>
<tr>
<td>99</td>
<td>StopPx</td>
<td></td>
<td>Price</td>
<td>Required for Stop, StopLimit orders</td>
</tr>
<tr>
<td>20020</td>
<td>NoRoutingCandidates</td>
<td></td>
<td>Number in repeating group</td>
<td>Repeating group for smart order routing, including all Candidate* fields below</td>
</tr>
<tr>
<td>20021</td>
<td>CandidateSecurityId</td>
<td></td>
<td>int</td>
<td>SecurityID of the routing candidate</td>
</tr>
<tr>
<td>20022</td>
<td>CandidateAccountId</td>
<td></td>
<td>int</td>
<td>AccountID of the routing candidate</td>
</tr>
<tr>
<td>20023</td>
<td>CandidateExchangeId</td>
<td></td>
<td>int</td>
<td>ExchangeId of the candidate</td>
</tr>
<tr>
<td>20024</td>
<td>CandidateExchangeName</td>
<td></td>
<td>String</td>
<td>ExchangeName of the candidate (id or name must be specified)</td>
</tr>
<tr>
<td>957</td>
<td>NoStrategyParameters</td>
<td></td>
<td>Number in repeating group</td>
<td>Repeating group for algo orders, including all StrategyParameter* fields below</td>
</tr>
<tr>
<td>958</td>
<td>StrategyParameterName</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>959</td>
<td>StrategyParameterType</td>
<td></td>
<td>INT</td>
<td>Always String(14)</td>
</tr>
<tr>
<td>960</td>
<td>StrategyParameterValue</td>
<td></td>
<td>String</td>
<td>Value of the parameter</td>
</tr>
</tbody>
</table>

### 22.4.7. Order Cancel Request message

Used to cancel an order. Sent from external client to AlgoTrader.

### Table 22.8. Order Cancel Request

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>StandardMsgType</td>
<td></td>
<td>Char</td>
<td>F</td>
</tr>
<tr>
<td>41</td>
<td>OrigClOrdID</td>
<td></td>
<td>String</td>
<td>ClOrdId to be cancelled</td>
</tr>
<tr>
<td>37</td>
<td>OrderID</td>
<td></td>
<td>String</td>
<td>orderID to be cancelled (assigned by AT)</td>
</tr>
</tbody>
</table>
22.4.8. Execution Report message

Used to receive order update information from AlgoTrader, such as confirmation for fills, cancellations, modifications or rejections. Sent from AlgoTrader to the external client.

Table 22.9. Execution Report

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>ClOrdID</td>
<td>Y</td>
<td>String</td>
<td>Unique ID of this particular request</td>
</tr>
<tr>
<td>22.4.8. Execution Report message</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 22.9. Execution Report

<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Account</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader account ID</td>
</tr>
<tr>
<td>17</td>
<td>ExecID</td>
<td>Y</td>
<td>String</td>
<td>Unique identifier for this execution report, UUID format</td>
</tr>
<tr>
<td>11</td>
<td>ClOrdID</td>
<td>Y</td>
<td>String</td>
<td>Client order ID this exec report is related to</td>
</tr>
<tr>
<td>37</td>
<td>OrderID</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader order ID</td>
</tr>
<tr>
<td>41</td>
<td>OrigClOrdID</td>
<td>C</td>
<td>String</td>
<td>Previous order identifier, equal to previous ClOrdID and sent after modification requests</td>
</tr>
<tr>
<td>38</td>
<td>OrderQty</td>
<td>Y</td>
<td>String</td>
<td>Total order quantity</td>
</tr>
<tr>
<td>6</td>
<td>AvgPx</td>
<td>Y</td>
<td>Price</td>
<td>Average price of all fills, 0 if no executions yet</td>
</tr>
<tr>
<td>151</td>
<td>LeavesQty</td>
<td>Y</td>
<td>Qty</td>
<td>Remaining quantity in the market. If order is still in the market LeavesQty = Qty - CumQty, 0 otherwise</td>
</tr>
<tr>
<td>14</td>
<td>CumQty</td>
<td>Y</td>
<td>Qty</td>
<td>Filled quantity so far</td>
</tr>
<tr>
<td>167</td>
<td>SecurityType</td>
<td>N</td>
<td>String</td>
<td>Type of security</td>
</tr>
<tr>
<td>48</td>
<td>SecurityID</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader Security ID</td>
</tr>
<tr>
<td>55</td>
<td>Symbol</td>
<td>Y</td>
<td>String</td>
<td>AlgoTrader Security Symbol</td>
</tr>
<tr>
<td>59</td>
<td>TimeInForce</td>
<td>Y</td>
<td>Char</td>
<td>How long an order remains active. Supported: 0, 1, 2, 3, 4, 6, 7</td>
</tr>
<tr>
<td>54</td>
<td>Side</td>
<td>Y</td>
<td>Char</td>
<td>1 or 2 for Buy/Sell</td>
</tr>
<tr>
<td>150</td>
<td>ExecType</td>
<td>Y</td>
<td>Char</td>
<td>Indicates the reason for sending this Execution Report</td>
</tr>
</tbody>
</table>

- 0: NEW, sent as a confirmation that the order has been accepted
- 6: PENDING CANCEL, received, but not processed yet
<table>
<thead>
<tr>
<th>Tag #</th>
<th>Field Name</th>
<th>Required</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
</table>
| 39    | OrdStatus  | Y        | Char      | Current order status
|       |            |          |           | - 0: NEW, order submitted
|       |            |          |           | - 1: PARTIALLY FILLED, order partially executed
|       |            |          |           | - 2: FILLED, order executed
|       |            |          |           | - 4: CANCELED, order canceled
|       |            |          |           | - 6: PENDING CANCEL, order will be cancelled
|       |            |          |           | - 8: REJECTED, order rejected
|       |            |          |           | - A: PENDING_NEW, order received, but not submitted yet
|       |            |          |           | - E: PENDING REPLACE, order will be replaced
| 58    | Text       | C        | String    | Additional information. Typically contains the details in case of rejection
Chapter 23.

Adapters

The following sections give a detailed overview of the different adapters available for AlgoTrader.

23.1. Fix Interface

AlgoTrader uses QuickFix/J\(^1\) for its Fix connections and currently supports FIX 4.2, 4.4 and 5.0. Because FIX messages are not compatible between different versions, the three distinct services Fix42OrderService, Fix44OrderService and Fix50OrderService exist. Incoming messages are handled by their corresponding FixXXMessageHandler classes.

To configure a Fix trading connection the following steps have to be taken care of:

- Add the corresponding fix trading profile to the VM argument `spring.profiles.active` (e.g. `cNXFix`):
  
  ```
  -Dspring.profiles.active=live,pooledDataSource,cNXFix,embeddedBroker,html5,InfluxDB
  ```

- Add the fix session to `/algotrader/bootstrap/conf/src/main/resources/fix.cfg` (Use the `fix-template.cfg` file as basis - do not delete the default section):
  
  ```
  [session]
  SessionQualifier=CNXT
  BeginString=FIX.4.4
  SenderCompID=xxx
  TargetCompID=CNX
  SocketConnectHost=dret-fix-ssl.currenex.com
  SocketConnectPort=443
  SocketUseSSL=Y
  Username=xxx
  Password=xxx
  ValidateIncomingMessage=N
  ResetOnLogon=Y
  Inactive=Y
  ```

- Make sure there is an entry in the MySQL account table where the column `ORDER_SERVICE_TYPE` matches the type of the fix interface (e.g. `CNX_FIX`), the column `SESSION_QUALIFIER` matches the `SessionQualifier` specified in the file `fix.cfg` and the `ACTIVE` column is set to 1.

If market data is also received through a Fix interface the following items need to be added as well:

- Add the corresponding fix market data profile to the VM argument `spring.profiles.active` (e.g. `cNXFix`):

---

\(^1\) https://www.quickfixj.org/
• Add the fix session to /algotrader/bootstrap/conf/fix.cfg (an example file fix-template.cfg is provided in the same directory):

```
[session]
SessionQualifier=CNXMD
BeginString=FIX.4.4
SenderCompID=xxx
TargetCompID=CNX
SocketConnectHost=dret-fix-ssl.currenex.com
SocketConnectPort=443
SocketUseSSL=Y
Username=xxx
Password=xxx
ValidateIncomingMessage=N
ResetOnLogon=Y
Inactive=Y
```

• When making subscriptions add the AdapterType corresponding to the Fix interface (e.g. CNX)

**Important**

Please make sure to have the setting `Inactive=Y` in both trading and market-data sections. Without this setting the fix session will be initialized before the remaining system has been fully initialized and might cause either trading or market data to malfunction.

For further information regarding QuickFix/J configuration please visit the [QuickFix/J documentation](https://www.quickfixj.org/usermanual/1.6.4//usage/configuration.html)

Per default Fix interfaces uses the following items to identify a particular instrument:

**Options**

- Exchange `IB_CODE`
- Option `TRANSACTION_CURRENCY`
- SecurityFamily `SYMBOL_ROOT`
- Option `STRIKE`
- Option `TYPE`
- Option `EXPIRATION`
Option CONTRACT_SIZE

Future
  Future TRANSACTION_CURRENCY
  Exchange IB_CODE
  SecurityFamily SYMBOL_ROOT
  Future EXPIRATION
  Future CONTRACT_SIZE

Forex
  Forex TRANSACTION_CURRENCY
  Exchange IB_CODE
  Forex BASE_CURRENCY

Stock
  Stock TRANSACTION_CURRENCY
  Exchange IB_CODE
  Stock SYMBOL

Fund
  Fund TRANSACTION_CURRENCY
  Exchange IB_CODE
  Index SYMBOL

23.1.1. FIX configuration

All FIX configurations are stored in fix.cfg file by default.

The file fix-template.cfg contains default parameters suggested by AlgoTrader for all FIX sessions. The individual [session] blocks should be added after the [default] block.

```plaintext
[default]
ConnectionType=initiator
HeartBtInt=30
ReconnectInterval=5
FileStorePath=files/fix
FileLogPath=log
FileLogHeartbeats=N
FileIncludeMilliseconds=Y
```
Details of individual FIX sessions are expected to be provided by the brokerages.

For further information regarding QuickFix/J configuration please visit the QuickFix/J documentation.

### 23.1.2. FIX logging

In addition to stock QuickFix/J configuration capabilities AlgoTrader provides a custom option to select a logging back-end out of those supported by QuickFix/J per individual session through custom `Logimpl` parameter.

Supported parameter values are:

- **file (default)**
  - Log to QuickFix/J standard file logger.
  
  The file option (which is the default) will create fix log files in the log sub-directory of the directory where AlgoTrader was started. One log file will be created for messages and one for QuickFix/J internal events. The name of both files contains the full FIX session name. To use the file logger the following settings are required within the `fix.cfg` file:

```plaintext
FileLogPath=log
FileLogHeartbeats=N
FileIncludeMilliseconds=Y
FileIncludeTimeStampForMessages=Y
```

- **none**
  - Disable logging. No FIX session events or messages will be logged.
  
  The none option might be especially useful for volume intensive market data sessions where persistent message log could be unnecessary or even excessive. Custom Fix logging options can be configured as follows:

```plaintext
[symbol]
SessionQualifier=FIXMD
BeginString=FIX.4.4
... 
LogImpl=none
```

- **screen**
  - Logs all QuickFix/J events and messages to the standard console logger.

---

3. [https://www.quickfixj.org/usermanual/1.6.4//usage/configuration.html](https://www.quickfixj.org/usermanual/1.6.4//usage/configuration.html)
**slf4j**

Log to the Simple Logging Facade for Java (SLF4J). Log entries will be committed to the logging back-end configured by SLF4J.

The slf4j allows log messages to be re-directed to the Log4J *Chapter 30, Logging* system which is highly configurable. Here is an example:

```plaintext
[session]
SessionQualifier=FIXMD
BeginString=FIX.4.4
...
LogImpl=slf4j
```

Log messages are logged to the following 4 loggers per default:

- `quickfixj.event` for regular events (e.g. sessions logging on)
- `quickfixj.errorEvent` for error events (e.g. connection issues)
- `quickfixj.msg.incoming` for incoming messages
- `quickfixj.msg.outgoing` for outgoing messages

A typical log statement would then look like this

```plaintext
```

It is also possible to customize the log categories per session for added flexibility. It is for example possible to log FIX messages from different FIX sessions into separate Files. The following example has two sessions, one for market data (FIXMD) and one for trading (FIXORD).

```plaintext
[session]
SessionQualifier=FIXMD
BeginString=FIX.4.4
...
LogImpl=slf4j
SLF4JLogIncomingMessageCategory=quickfixj.msg.md.incoming
SLF4JLogOutgoingMessageCategory=quickfixj.msg.md.outgoing

[session]
SessionQualifier=FIXORD
BeginString=FIX.4.4
...
LogImpl=slf4j
```
By adding the following sections to the log4j2.xml separate log files for market data (fix-md.log) and trading (fix-ord.log) will be created.

```xml
...<RollingFile
    name="FixMD"
    fileName="log/fix-md.log"
    filePattern="log/fix-md-%d{MM-dd-yyyy}-%i.log.gz">
    <PatternLayout pattern="%d %m %n"/>
    <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
    </Policies>
</RollingFile>
<RollingFile
    name="FixORD"
    fileName="log/fix-ord.log"
    filePattern="log/fix-ord-%d{MM-dd-yyyy}-%i.log.gz">
    <PatternLayout pattern="%d %m %n"/>
    <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
    </Policies>
</RollingFile>
...<logger name="quickfixj.msg.md" level="info" additivity="false">
    <AppenderRef ref="FixMD"/>
</logger>
<logger name="quickfixj.msg.ord" level="info" additivity="false">
    <AppenderRef ref="FixORD"/>
</logger>

For both files a daily rolling file scheme is used where at the end of a day all contents of each file will be zipped (e.g. fix-md-12-08-2019.log.gz). In addition if the file size reaches 250MB a new zip file will be created as well.

23.1.3. FIX message persistence

One can use standard QuickFix/J facilities to store FIX messages either in a local file or in a relational database through JDBC DataSource interface. For details please refer to the QuickFix/J documentation. For further details please visit the QuickFix/J documentation

4 https://www.quickfixj.org/usermanual/1.6.4//usage/configuration.html#Storage
23.1.4. FIX Drop-copy support

LMAX and Trading Technologies interfaces provide support for so called drop-copy mode wherein the adapter can receive order status and fill messages from orders initiated externally (usually by external applications such as native trading front-ends). By default external fills get recorded as transactions of the SERVER strategy and allocated to the external account specified in the original execution report message. One, however, can provide a custom implementation of DropCopyAllocator interface in order to apply custom transaction allocation logic.

23.2. Crypto Exchange interfaces

AlgoTrader provides several crypto currency exchange adapters which are based on REST, Web Socket or REST.

When trading crypto currencies it is recommended to update the following properties inside conf.properties. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```
# the currency all portfolio balances will be calculated in
misc.portfolioBaseCurrency=BTC

# the number of digits all portfolio balances will be displayed with
misc.portfolioDigits=8
```

This will cause all balances to be displayed in BTC with a precision of 8 digits

23.2.1. Custom currency mapping

Not all exchanges use the same names for the same coins (e.g. BTC or XBT for Bitcoin). So that the system recognises it’s the same instrument and trades it properly across exchanges, there is a mapping algoritrader/bootstrap/conf/src/main/resources/currency-code-mappings.csv. If the matching entry exists in that file (adapter code, exchange code and adapter currency code), then the defined AlgoTrader currency code will be used throughout the system and the adapter specific name will be used while communicating with the exchange.

23.2.2. Crypto-Order Constraints

All securities contain fields which describe valid order quantities and prices.

Crypto reference data services set these values provided this information is made available by the exchange. The crypto-adapters validate orders versus these constraints and reject them before sending them to the exchanges/brokers.

The constraints are defined by:

- MinQty - minimum order quantity
- MaxQty - maximum order quantity
- QtyIncr - quantity increment
- **MinPrice** - minimum price
- **MaxPrice** - maximum price
- **PriceIncr** - price increment
- **MinNotional** - minimum notional value of the order. For example for limit order: Quantity * Limit price

Securities are updated automatically if there is a change on the exchange side. To change this behaviour (not to override those constraints), the user has to set the corresponding property inside the adapter specific property file (e.g. inside the `conf-bmx.xml` file for the BMX adapter). Alternatively the properties can be changed via Section 2.4, “VM Options”:

```json
#{"type":"Boolean","label":"if true override existing security families fields..."}
bmx.overrideSecurityFamilies = true
```

Check the corresponding crypto-adapter sections below on what constraints are validated for each broker/exchange.

### 23.2.3. Supported Crypto-Security Types

The following table contains the tradable security types per adapter

**Table 23.1. Tradable security types**

<table>
<thead>
<tr>
<th>Broker/Exchange</th>
<th>Spot</th>
<th>Perpetual swaps</th>
<th>Futures</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B2C2</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Binance</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Bitfinex</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>BitFlyer</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>BitHumb Pro</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>BitMex</strong></td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td><strong>BitStamp</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Bittrex</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Blockfills</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Coinbase Pro</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>CryptoFinance</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Deribit</strong></td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><strong>Enigma</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>eToroX</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Gemini</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
## 23.2.4. Supported Crypto-Order Types

The following table contains the valid order types per adapter.

<table>
<thead>
<tr>
<th>Broker/Exchange</th>
<th>Spot</th>
<th>Perpetual swaps</th>
<th>Futures</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huobi Spot</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Kraken Spot</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>LMAX Digital</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>OKEx</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>OKCoin</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

### Table 23.2. Order type constraints

<table>
<thead>
<tr>
<th>Broker/Exchange</th>
<th>Exchange Trading</th>
<th>Margin Trading</th>
<th>Market</th>
<th>Limit</th>
<th>Stop</th>
<th>Stop Limit</th>
<th>Previously Indicated</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2C2</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Binance</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Bitfinex</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>BitFlyer</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>BitHumb Pro</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>BitMEX</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>BitStamp</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Bittrex</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Blockfills</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Coinbase Pro</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>CryptoFinance</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Deribit</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Enigma</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>eToroX</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Gemini</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Huobi Spot</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Kraken Spot</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>LMAX Digital</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>OKEx</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
### 23.2.5. Supported Crypto Adapter Account Functionality

The following table contains the account adapter functionality as available per adapter.

<table>
<thead>
<tr>
<th>Adapter</th>
<th>Withdrawals</th>
<th>Deposit Retrieval</th>
<th>Address Retrieval</th>
<th>Account Events</th>
<th>Account Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B2C2</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Binance</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Bitfinex</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>BitFlyer</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>BitHumb Pro</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>BitMex</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>BitStamp</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>BitTrex</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Blockfills</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Coinbase Pro</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>CryptoFinance</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Deribit</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Enigma</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>eToroX</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Gemini</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Huobi Spot</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Kraken Spot</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>LMAX Digital</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>OKEx</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>OKCoin</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 23.3. Adapter Rate Limits

Most exchange APIs have rate-limits. It means that there's a limited amount of requests that can be made by single user in given timespan. Exceeding the rate-limit may cause user requests to be temporarily blocked. In
order to ensure stable connectivity with the exchange by, AlgoTrader outgoing HTTP calls are limited. Request rate is configurable by properties.

```java
xyz.rateLimits = 100/1s, 1000/1m
```

The above example means that adapter xyz allows 100 requests per second, but no more than 1000 requests per minute.

Rate-limit property should be specified in the following format:

```
[number of requests in timespan]/[duration][time unit]
```

Valid time units are:

- s (seconds)
- m (minutes)
- h (hours)
- d (days)

Rate-limit examples:

- 10/1s (10 requests per second)
- 100/2m (100 requests per 2 minutes)

---

**Note**

AlgoTrader comes with predefined rate-limit values and there is no need to override them unless the exchanges have changed their limits or you have more restrictive requirements.

### 23.4. Session life-cycle events

All trading interface adapters generate session events, which enable the server engine as well as individual strategies to listen for and react to session events such as session being fully established or temporary loss of connectivity.

```java
@override
public void onSessionEvent(final SessionEventVO event) {
    switch (event.getState()) {
        case CONNECTED:
```
Specially when connecting to a slow Market Data adapters it might be necessary to listen to those session events and only subscribe for securities once the session is in the `LOGGED_ON` state.

### 23.5. Automatic order reconciliation after re-connect

Most crypto exchanges provide REST and/or Web Socket APIs only. Unfortunately these protocols are stateless. So in case the AlgoTrader server gets disconnected from an exchange for some time or the AlgoTrader server is restarted, an active order may get cancelled or executed in the meantime.

For this purpose AlgoTrader provides an automatic order reconciliation feature. An automated order reconciliation process is invoked once the AlgoTrader server reconnects to synchronize the orders statuses between the exchange on the AlgoTrader server.

In case of trading adapters that use FIX interface, order reconciliation is provided automatically by the exchange FIX server. This is done using the `Resend` FIX protocol message. Unfortunately the `Resend` message is not supported by the Coinbase Pro exchange FIX server and BitStamp FIX server. Both adapters use AlgoTrader provided order reconciliation via Rest on their FIX session reconnect.

In case you want to turn off the AlgoTrader provided order reconciliation, add `noOrderReconciliation` into active Spring profiles list when starting the AlgoTrader server.

Please check the corresponding adapter section to see if the automatic order reconciliation is implemented for your exchange.

If your Strategy relies on reconciliation process and required information about whether it is currently running or not, it is possible to subscribe for ReconciliationEvents. After subscription, your Strategy will be notified about reconciliation process being started and finished. Code example below shows how to subscribe to such events from your StrategyService class:

```java
@Override
protected void onReconciliationEvent(final ReconciliationEventVO event) {
    // place code that reacts on reconciliation events here
}
```
One example use case that you can implement with such reconciliation events is state recovery processing after AlgoTrader was stopped and started (it is recommended to wait until reconciliation is finished and all order statuses are updated before starting your regular Strategy process)

23.6. Bloomberg

The Bloomberg adapter supports Market Data, Historical Data as well as Reference Data.

For a related trading adapter see Section 23.7, “Bloomberg Tradebook”.

The Bloomberg adapter provides both synchronous connections and asynchronous connections. Asynchronous connections are generally used for live market data whereas synchronous connections are used for retrieval of historical data as well as retrieval of reference data.

If market data is received through the Bloomberg interface the following items need to be added:

- **Add the profile** `bBMarketData` **to the VM argument** `spring.profiles.active`:

  ```
  `spring.profiles.active=live,pooledDataSource,bBMarketData,embeddedBroker,html5,InfluxDB`
  ```

- **When making subscriptions add the AdapterType BB**

  Bloomberg uses the **BBGID** field of the Security table to identify instruments.

For further details on the Bloomberg interface please visit the [Bloomberg Open API](https://www.bloomberg.com/professional/support/api-library/)

23.7. Bloomberg Tradebook

*Bloomberg Tradebook* provides trading access to traditional asset classes and native execution algos.

For a related market data adapter see Section 23.6, “Bloomberg”.

AlgoTrader Tradebook adapter allows access to Bloomberg Tradebook via FIX API. Execution algo orders, 'benchmark strategy orders' as they are called by Bloomberg can be used by the usual `OrderService` methods with custom FIX tags specified using order object properties (see Section 17.2.3, “Order Properties”). Tradebook adapter has convenience Tradebook adapter specific methods available for TWAP, VWAP and Go Along benchmark strategy orders.

The following example shows how to send Tradebook VWAP order:

```java
((TDBOrderServiceImpl)
orderService.getExternalOrderService(AdapterType.TDB)).sendTradebookVWAPOrder(...);
```

---

5 [https://www.bloomberg.com/professional/support/api-library/](https://www.bloomberg.com/professional/support/api-library/)

6 [https://www.bloomberg.com/professional/solution/tradebook/](https://www.bloomberg.com/professional/solution/tradebook/)
23.8. Currenex

The main features of the Currenex platform are

• Live tradable rates
• Liquidity in all the major currency pairs
• Straight through processing of order executions

The Currenex implementation of the FIX/4.4 protocol has some peculiarities

• Requires additional acknowledgement `TradingSessionStatus` message indicating the trading session is fully initialized
• Supports only subset of standard Order Expiry (Time in Force) types
• Uses `FOREX_MARKET` (type C) and `FOREX_LIMIT` (type F) for Market and Limit orders

Currenex uses the columns `Forex BASE_CURRENCY` and `SecurityFamily CURRENCY` to identify an instrument.

23.9. DukasCopy

AlgoTrader supports both trading and market data connectivity via broker DukasCopy.

Since the DukasCopy FIX protocol implementation does not follow the FIX standard very closely, we at the moment don’t support order modifications via Dukascopy trading adapter. Modification of an open order can be achieved by cancelling and submitting a new order.

Note that Dukascopy does not support Stop Limit orders.

23.10. Exante (XNT)

Exante brokerage adapter, FIX 4.4 protocol based.

The adapter supports market data and trading.

Exante provides its customers with a demo platform with the same characteristics as the live platform. Market data in the Demo platform is delayed by at least 30 minutes. Live platform requires setting up an SSH tunnel, details for which are available at their website.

Reference data download is currently not supported by AlgoTrader. Individual securities may be enabled on AlgoTrader side for use with Exante by setting their `XNTID` database field with the correct Exante instrument code.

23.11. EzeSoft / Real Tick

EzeSoft / RealTick provides connectivity to about 30 institutional and 10 retail brokers.
The EzeSoft / RealTick Fix interface currently supports only Order Processing.

The Fix Implementation of EzeSoft / RealTick is well conforming with the Fix Standard no customizations had to be made.

The Fix interface uses standard Fix instrument definitions mentioned at the end of section Section 23.1, “Fix Interface”.

23.12. Fortex

Fortex uses almost vanilla Fix/4.4 protocol with very few customizations. It supports FX only.

- Supports GTC, IOC and FOK time-in-force parameters only
- Requires filled quantity to be included in order cancellation messages

Fortex uses the columns Forex BASE_CURRENCY and SecurityFamily CURRENCY to identify an instrument.

23.13. FXCM

FXCM interface FIX/4.4 protocol does not deviate much from the standard but has some peculiarities about the way FIX sessions are established

- Unlike many other FIX connectivity providers who provide separate FIX sessions for market data and trading interfaces FXCM by default offers one session for both market data feed and trading operations
- Uses extra UserRequest / UserResponse message exchange to authenticate the user and to fully initialize the session

FXCM uses the columns Forex BASE_CURRENCY and SecurityFamily CURRENCY to identify an instrument.

23.14. IB Native Interface

The native IB Interface connects to the local Trader Workstation (TWS) or IB Gateway instance and uses methods supplied by the IB client. The interface is fully capable of handling IB's Financial Advisor functionality like Sub Accounts, Account Groups and Allocation Profiles.

The IB interface supports Market Data, Historical Data, Order Processing, Retrieval of account information as well as Reference Data.

Note

You get market data for a minimum of 100 instruments with subscriptions (depends on your commissions and assets with IB). You can buy up to 10 quote boosters for USD 30 each, which provide 100 additional instruments each (max 1000). For details, consult the IB market data fees.

Similar restrictions/extensions exist for historical data. Those details can be viewed here:8

To configure an IB connection the following steps have to be taken care of:

• Add the profile iBNative to the VM argument spring.profiles.active:

```- Dspring.profiles.active=live,pooledDataSource,iBNative,embeddedBroker,html5,InfluxDB```

• Make sure there is an entry in the account database where the column ORDER_SERVICE_TYPE is set to IB_NATIVE).

If market data is also received through the IB interface the following items need to be added as well:

• Add the profile iBMarketData to the VM argument spring.profiles.active:

```- Dspring.profiles.active=live,pooledDataSource,iBMarketData,embeddedBroker,html5,InfluxDB```

• When making subscriptions add the AdapterType IB

The IB interface has the following options to identify a particular instrument:

• CONID specified in the security table

• Use instrument symbols and additional data depending on the instrument type:

**Options**

- Exchange IB_CODE
- SecurityFamily CURRENCY
- SecurityFamily SYMBOL_ROOT
- Option STRIKE
- Option TYPE
- Option EXPIRATION
- SecurityFamily CONTRACT_SIZE

**Future**

- SecurityFamily CURRENCY
- Exchange IB_CODE

---

8 http://interactivebrokers.github.io/tws-api/historical_limitations.html#hd_availability
SecurityFamily SYMBOL_ROOT
Future EXPIRATION
SecurityFamily CONTRACT_SIZE

Stock
SecurityFamily CURRENCY
Exchange IB_CODE
Stock SYMBOL

Index
SecurityFamily CURRENCY
Exchange IB_CODE
Index SYMBOL

Combination
SecurityFamily CURRENCY
Exchange IB_CODE
SecurityFamily BASE_SYMBOL
Security CONID of each Component
Component QUANTITY

In addition the following items apply to the IB Native interface

• The IB Native interface uses the RT_VOLUME⁹ events to process incoming trade events

• The IB Native interface propagates daily OPEN and CLOSE prices to strategies in case the following property inside conf-ib.properties is enabled. Alternatively the properties can be changed via Section 2.4, “VM Options”

```bash
# enables emission of generic open and close ticks
ib.emitOpenClose = true
```

• The IB Native interface propagates VWAP prices to strategies in case the following property inside conf-ib.properties is enabled. Alternatively the properties can be changed via Section 2.4, “VM Options”

```bash
# enables emission of generic VWAP ticks
ib.emitVWAP = true
```

⁹ https://interactivebrokers.github.io/tws-api/tick_types.html#rt_volume
• The IB Native interface expects orders to be sent with their order ids in ascending order. The Class IBOrderIdSynchronizer is responsible to make sure order ids are actually in ascending order. In case an order id is skipped the IBOrderIdSynchronizer will wait for up to maxOrderSyncTime milliseconds for the order with the correct order id to arrive.

• The IB Native interface supports trading of tradable / non-synthetic combinations by placing BAG orders through the IB interface.

• The IB Native interface reports volBid, volAsk and vol in lots of 100 contracts for US equities. Please see the following page for further details on handling of Odd Lot Orders\textsuperscript{10}

For further details on the IB native interface please visit the IB API Reference Guide\textsuperscript{11}

\subsection{23.14.1. IB Market Data Subscriptions}

In the traditional financial sector (excluding cryptocurrencies) market data is not free and requires market data subscriptions.

IB provides free 15min delayed data. It is possible to obtain this free 15min delayed data when logged in via a trial account. This however is available only when logged in via Trader Workstation (TWS). Please see Section 23.14.2, “Delayed IB Market Data”

Market data can be accessed both through the IB paper trading account as well as the live trading account.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{note.png}
\caption{Note}
\end{figure}

To get a market data subscriptions one has to login to the IB account management\textsuperscript{12} with the live trading account. Then follow these steps:

\begin{itemize}
\item The paper trading account has one single username assigned to it. The live trading account can have multiple user names.
\item For each username (live account & paper trading account) only one session can exist at the same time. If you login with the same username on a different machine the other session will get logged out.
\item If the live account username (that is sharing its market data subscription with the paper trading account) is currently logged in, the paper trading account doesn’t get market data until the live account is again logged out.
\item If a client wants to login to the live trading account at the same time that AlgoTrader is connected to the paper trading account, he has to create a second username under the live account and purchase additional market data subscriptions for that username.
\end{itemize}

\begin{footnotes}
\item[10] \url{https://ibkr.info/node/1062}
\item[11] \url{https://www.interactivebrokers.com/en/software/api/api.htm}
\item[12] \url{https://gdcdyn.interactivebrokers.com/sso/Login}
\end{footnotes}
1. Select Settings / User Settings in the menu on the left. Then select Market Data Subscriptions on the right.

2. Then click on the icon next to *Current Subscriptions*.

3. Then select the region (e.g. *North America*).

4. On the next screen individual market data subscriptions can be selected.

---

**Figure 23.1. Market Data Subscriptions 1**

**Figure 23.2. Market Data Subscriptions 2**
Typical market data subscriptions are:

- **IDEAL FX**: free Forex market data
- **NASDAQ (Network C/UTP)**: live market data for NASDAQ listed equities
- **NYSE (Network A/CTA)**: live market data for NYSE listed equities
- **US Securities Snapshot and Futures Value Bundle**: live market data for US futures and snapshot data for US equities (AT cannot process snapshot data, so in addition NASDAQ and NYSE has to be subscribed as well)
To use these market data subscriptions through the paper trading account follow these steps:

1. Select Settings / User Settings in the menu on the left. Then select Paper Trading Account on the right.

2. Then select Yes next to Share real-time market data subscriptions with paper trading account.

3. Then Select the username whose market data you want to share. This will share the market data subscriptions of the live account with the paper trading account.
23.14.2. Delayed IB Market Data

There are several prerequisites in order to use the free delayed Market Data from IB:

• TWS needs to be used. IB Gateway is not supported for free delayed market data

• Delayed market data subscriptions are available with trial accounts only. After starting the TWS, simply go to Return to the demo and enter your email address (see screen shot below)
• Similarly to IB Gateway, TWS should be configured to work with AlgoTrader: After logged in, go to Edit - > Global Configuration... -> API -> Settings. Make sure following settings are used: Enable ActiveX and Socket Clients should be enabled, Read-Only API should be disabled, Socket port should be set to 4001.

• The `ib.pricefeed.allowDelayedMarketData` property should be set to true (it is false by default)

![Login Simulated Trading](image)

Figure 23.7. Delayed IB Market Data

23.14.3. Custom functions in IB Native Account adapter

IB Native adapter’s specific `AccountService` has several additional functions compared to the standard `AccountService`. It provides access directly to Interactive Brokers.

Allocation Profile Retrieval

Allows a financial advisor to manually request an allocation profile’s configuration data.

```java
Map<String, Double> allocations = 
((IBNativeAccountService) accountService
 .getExternalAccountService("allocation profile")
 .getAllocations(allocationProfileName));
```
Custom functions in IB Native Account adapter

Accounts Retrieval
 Allows a financial advisor to retrieve the list of accounts in an account group.

```java
Set<String> accounts =
        ((IBNativeAccountService) accountService
            .getExternalAccountService(/*account group*/)
            .getAccounts();
```

Managed Accounts Retrieval
 Allows a financial advisor to retrieve the list of IB numbers of the managed client accounts.

```java
Set<String> managedAccounts =
        ((IBNativeAccountService) accountService
            .getExternalAccountService(/*account number*/)
            .getManagedAccounts();
```

Account positions retrieval
 Allows a user to manually request current securities' positions from IB.

```java
List<SecurityPositionVO> positions =
        ((IBNativeAccountService) accountService
            .getExternalAccountService(/*account number/account group/allocation profile*/)
            .retrieveAccountPositions(accountId);
```

Allocation Profile Update
 Allows a financial advisor to update an allocation profile.

```java
((IBNativeAccountService) accountService.getExternalAccountService(/*allocation profile name*/)
    .updateAllocationProfile(Profile profile, int type);`
23.14.4. IB Generic Tick Events

In addition to Section 18.6, "Generic Events" IB adapter also provides Generic Ticks (class IBGenericTickVO), unlike GenericEvents, IBGenericTicks are IB specific. A Generic Tick Event represents additional price information on a particular instrument made available by market data provider (e.g. open price, close price, vwap price).

As IBGenericTickVO is a subclass of MarketDataEventVO a strategy will automatically get Generic Tick Events delivered when it has subscribed to the corresponding instrument.

A Generic Tick has a TickType which can be one of OPEN, HIGH, LOW, CLOSE, SETTLEMENT, OPEN_INTEREST, IMBALANCE or VWAP. A Generic Tick Event can hold either a BigDecimal, Double or Integer value.

23.15. IB Fix Interface

The IB Fix Interface provides the same Order Management features as the IB Native Interface. However Market Data is not available through this interface.

The interface is fully capable of handling IB's Financial Advisor functionality like Sub Accounts, Account Groups and Allocation Profiles.

For further details on the IB Fix interface please visit the IB FIX/CTCI Users' Guide13

The IB Fix interface uses standard Fix instrument definitions mentioned at the end of section Section 23.1, "Fix Interface".

23.16. Intrinio Dividend feed

Intrinio14 provides different types of market data. As of now AlgoTrader only supports the dividends data feed.

It can be enabled by activating the profile iNTRDividendGenericEvents and by setting the following property in conf-intr.properties:

intr.apiKey = XXXX

23.17. JP Morgan

The JP Morgan Fix interface supports Order Processing only.

As the JP Morgan Fix Implementation is well conforming with the Fix Standard no customizations had to be made

The JP Morgan Fix interface uses standard Fix instrument definitions mentioned at the end of section Section 23.1, “Fix Interface”.

---

14 https://intrinio.com/
23.18. LMAX Global

Supports only a limited number of securities, mainly Forex.

LMAX Global implementation of the FIX/4.4 protocol has some peculiarities:

- Uses predefined contract modifiers for market data events and order quantities. The contract modifiers are not included in FIX messages and have to be applied by the interface adapter.
- Uses custom message dictionary
- Supports **DAY**, **GTC**, **IOC** and **FOK** time-in-force parameters for market and limit orders.
- Supports only **DAY** and **GTC** time-in-force parameters for stop orders (stop and stop-limit).
- supports trading status signaling temporary suspension and resumption in trading of individual securities.

LMAX Global uses the column **LMAXID** of the security table to identify an instrument

23.19. Nexus Prime

*Nexus Prime*\(^{15}\) is a MetaTrader MT4 FIX interface provided by IS Risk Analytics. The Nexus Prime interface uses Fix 4.4 and it supports FX only. Due to the underlying MetaTrader MT4 a few limitations apply.

- Market Data subscriptions cannot be cancelled
- Orders cannot be modified, instead one needs to cancel the current order first and then resend a new one.
- Buy limit orders need to be placed below the market price. Sell limit orders need to be placed above the market price.
- Buy stop orders need to be placed above the market price. Sell stop orders need to be placed below the market price.
- Minimum trade size allowed on most currency pairs is .01 lots which is 1000 notional

Nexus Prime uses the columns **Forex BASE_CURRENCY** and **SecurityFamily CURRENCY** to identify an instrument.

23.20. One Zero

One Zero provides connectivity to a number of exchanges. There are differences in supported symbols between the exchanges. Pair symbols can be mapped to the expected form using currency-code-mappings.csv file.

OneZero Financial Systems LLC develops low-latency software systems. The Company provides traders, managers, and brokers with software tools to identify market opportunities, automate their trading, and control risk.

\(^{15}\) [https://www.isriskanalytics.com/trading-technology/](https://www.isriskanalytics.com/trading-technology/)
One Zero implements the FIX/4.4 protocol and requires to use TLSv1:

```
EnabledProtocols=TLSv1
```

### 23.21. PrimeXM

The PrimeXM FIX/4.4 interface implementation follows the Fix Standard closely, but uses MassQuote messages for conveying the market data. Each MassQuote message has to be acknowledged by the FIX client.

Only Forex instruments are supported by the PrimeXM Fix Interface.

Market, limit and stop orders are supported but only with time in force IOC (Instant or Cancel) or FOK (Fill or Kill).

Order modifications are not supported.

### 23.22. Quandl

Quandl\(^{16}\) is a public service that provides a wide range of financial, economic and alternative data. It is mostly end of day data but also some intra-day (e.g. hourly) data. To find out if they have what you are looking for, check their data products page.\(^{17}\) AlgoTrader allows downloading historical data from Quandl. For more information about Quandl please have a look at the Quandl Docs/Help.\(^{18}\)

Data on Quandl is divided into databases. Each database contains multiple datasets. For instance EOD database contains end-of-day data for all publicly-traded US stocks. Each database/dataset pair is uniquely identified by database_code/dataset_code pair. For instance EOD/AAPL is the globally unique code for the AAPL stock dataset within the EOD database. The Quandl database browser\(^{19}\) can be used to find suitable databases for desired instrument type, region and data type.

The `QdlHistoricalDataService` is integrated with the AlgoTrader Historical Data Download and needs to be enabled by specifying the `qdlHistoricalData` Spring profile (see section Section 19.3, “Historical Data Download”). The `QdlHistoricalDataService` transforms retrieved Quandl data into AlgoTrader bars. Transformation rules between the Quandl data format and AlgoTrader Bar format are defined in the file `quandl.yml`. By default the file `quandl.yml` already contains the transformation rules for most commonly used Quandl databases. Additional transformation rules can be added to the file as needed:

```yaml
EOD:

   barSize: DAY_1
   columnMapping:
     dateTime: Date
     open: Open
     high: High
```

\(^{16}\) [https://www.quandl.com/](https://www.quandl.com/)

\(^{17}\) [https://www.quandl.com/search?query=](https://www.quandl.com/search)

\(^{18}\) [https://www.quandl.com/docs-and-help](https://www.quandl.com/docs-and-help)

\(^{19}\) [https://www.quandl.com/search](https://www.quandl.com/search)
The Quandl database code

barSize supported by the Quandl database (e.g. DAY_1 or MIN_1)

Column mappings between Quandl data fields and AlgoTrader BarVO fields

The relevant properties for the Quandl adapter are defined inside the file conf-qdl.properties where they can be changed. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```java
#{"type":"String","label":"API Key"}
qdl.apiKey = ATVxxxxxxxxxxxxx
```

To use the QdlHistoricalDataService please replace the property qdl.apiKey with the API Key that can be retrieved through the Quandl Account Settings.

In terms of historical data download a mapping between the Quandl database and the Security entity is defined by the quandl_database field in the security table. Similarly a mapping between the Quandl dataset and the Security entity is defined by the quandl_dataset field in the security table. AlgoTrader sample data files (samples/db/mysql/mysql-data.sql already contain quandl_database/quandl_dataset values for all sample security families and most sample securities.

### 23.23. QuantHouse

The QuantHouse adapter is based on the QuantHouse ultra low latency market data feed QuantFEED. The QuantHouse adapter supports live Market Data.

If market data is received through the QuantHouse interface the following items need to be added:

- Add the profile qHMarketData to the VM argument spring.profiles.active:

```bash
-Dspring.profiles.active=live,pooledDataSource,qHMarketData,embeddedBroker,html5,InfluxDB
```

- When making subscriptions add the AdapterType QH

QuantHouse uses the Exchange MIC and Security SYMBOL fields to identify instruments.

For further details on the QuantHouse interface please contact QuantHouse

### 23.24. Refinitiv Elektron (Reuters)

Refinitiv is a global provider of financial markets data. It supplies Reuters news and content.

---

20 https://www.quanthouse.com/
AlgoTrader supports market data using Websocket API. The availability of data depends on the account held. The securities need to be in the database and value of the field ‘RIC’ is used to communicate with Refinitiv.

The properties of the adapter are defined inside the file `conf-refinitiv.properties`. Alternatively they can be changed via Section 2.4, “VM Options”

```java
# Refinitiv ( Reuters ) properties
#
#{"type":"String","label":"Refinitiv Authentication Url"}
refinitiv.restUrl = https://api.refinitiv.com:443/auth/oauth2/beta1/token

#{"type":"String","label":"Refinitiv Discovery Url"}
refinitiv.restDiscoveryUrl = https://api.refinitiv.com/streaming/pricing/v1/

#{"type":"String","label":"Web Socket Url"}
refinitiv.wssUrl = wss://amer-3.pricing.streaming.edp.thomsonreuters.com:443

#{"type":"String","label":"API Key"}
refinitiv.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
refinitiv.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"The Client Id"}
refinitiv.clientId = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"ch.algotrader.config.RateLimit","label":"REST API Rate Limit"}
refinitiv.rateLimits = 60/1m

#{"type":"String","label":"Location of the endpoint to provide market data: eu, us, ap"}
refinitiv.endpointLocation = eu

#{"type":"Integer","label":"Number of maximum idle period between incoming messages in seconds"}
refinitiv.websocketTimeoutSeconds = 50
```

### 23.25. SocGen

The SocGen FIX/4.2 interface supports Order Processing only.

The SocGen Fix Implementation follows the Fix Standard closely, but some minor customizations according to the ‘SocGen FIX Rules of Engagement’ had to be made. Additionally exchange specific restrictions rules defining the allowed order type / TIF combinations were added.

---

21 https://developers.refinitiv.com/elektron/websocket-api
Only future instrument orders are supported by the SocGen Fix Interface.

23.26. Trading Technologies (TT)

Supports a wide range of future and option contracts tradable at multiple venues / exchanges.

- TT uses the column TTID of the security table to identify instruments
- Provides a reference data service that can be used to download contract definitions
- Supports drop-copy sessions

The relevant properties for TT adapter are defined inside the file conf-tt.properties where they can be changed. Alternatively the properties can be changed via Section 2.4, “VM Options”.

```java
#{"type":"String","label":"market request user name"}
tt.username = xxx
```

The tt.username property defines the username on the Trading Technologies side.

23.27. UBS

The UBS Fix interface supports Order Processing for futures and options only.

As the UBS Fix Implementation is well conforming with the Fix Standard no customizations had to be made.

The UBS Fix interface uses standard Fix instrument definitions mentioned at the end of section Section 23.1, “Fix Interface”.

23.28. B2C2

B2C2 is a cryptocurrency market maker/OTC trading services provider.

The B2C2 interface supports Market Data, Order Processing, Request for Quote (RFQ) process, Retrieval of account information as well as Reference Data. B2C2 only allows one RFQ to be in effect at a time and a second one invalidates the previous.

The relevant properties for the B2C2 adapter are defined inside the file conf-b2c2.properties where they can be changed. Alternatively the properties can be changed via Section 2.4, “VM Options”.

```java
#{"type":"Long","label":"The id of the account to be used in conjunction with the b2c2 RFQ (Request For Quote) process."}
b2c2.quoteAccountId = 207

#{"type":"String","label":"Price levels to subscribe per currency - coma separated key-value pairs"}
b2c2.priceLevels = BTC=10,ETH=50
```
B2C2 does not have an order book but will stream prices to you at the requested level.

The price levels define the quantity (in base currency) you wish to receive streamed prices for.

Above a certain value (to be agreed with B2C2), you will not be able to execute orders directly but need to go through the RFQ process.

In order to populate the database with B2C2 Accounts, Exchanges, Security Families and Securities run the ReferenceDataStarter with b2C2ReferenceData spring profile enabled and program argument: all. For further details please visit Chapter 20, Reference Data.

Note that time in force (TIF) for B2C2 Previously Indicated orders has to be Immediate or Cancel (IOC) or Fill or Kill (FOK), and for non RFQ orders only FOK is supported.

### 23.28.1. B2C2 Order Constraints

B2C2 supports previously indicated (for quote execution), limit and market orders.

#### Table 23.4. B2C2 constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>-</td>
</tr>
<tr>
<td>MaxQty</td>
<td>-</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>

### 23.29. Binance

*Binance*[^22] is a cryptocurrency exchange. Please see the *API reference*[^23] page for the technical details.

Binance provides Java library for interacting with Binance API. It supports REST requests to endpoint providing orders functionality, account data and reference data. Support for market data is done using WebSocket API.

The relevant properties for the Binance adapter are defined inside the file `conf-bnc.properties` where they can be changed. Alternatively the properties can be changed via *Section 2.4, “VM Options*:

```properties
#{"type":"String","label":"API Key"}
bnc.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
```

[^22]: https://www.binance.com/
[^23]: https://github.com/binance-exchange/binance-java-api
A Binance account is necessary in order to use Binance adapter. Unique apiKey and apiSecret settings must be set to the actual values (either in the properties file or by setting a VM argument).

Note
Binance is very time sensitive, i.e. if your computer is ahead of the Binance system clock, the API might reject your orders with an exception similar to

```java
com.binance.api.client.exception.BinanceApiException: Timestamp for this request was 1000ms ahead of the server's time
```

To prevent these issues, we suggest synchronizing your system clock with an internet reference time using e.g. this time sync tool\(^24\).

23.29.1. Binance Order Constraints

Binance supports exchange and margin trading.

AlgoTrader currently supports market, limit, stop and stop limit orders on Binance.

Note
Binance does not support order modifications.

Note that Binance has restrictions to the amount of (algo) orders that can be placed on an instrument or exchange. See the Binance API filter page\(^25\) for details.

Table 23.5. Binance constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>minimum order quantity</td>
</tr>
<tr>
<td>MaxQty</td>
<td>maximum order quantity</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>quantity increment</td>
</tr>
</tbody>
</table>

\(^{24}\) http://www.timesync tool.com/

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinPrice</td>
<td>minimum price</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>maximum price</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>price increment</td>
</tr>
<tr>
<td>MinNotional</td>
<td>minimum notional value</td>
</tr>
<tr>
<td>Details</td>
<td>Available constraints on Binance are listed here: Trading-Rule⁵⁶</td>
</tr>
</tbody>
</table>

Automatic order reconciliation (on WebSocket reconnect) is implemented for Binance.

### 23.29.2. Binance Account Management

#### Table 23.6. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>It is possible to withdraw BTC, ETH, LTC, NEO and BNB. The withdrawal address has to be provided in withdrawContext. A secondary address identifier for coins like XRP, XMR etc has to be provided as paymentId in the withdrawContext. The address description is optional.</td>
</tr>
<tr>
<td>Deposit</td>
<td>Addresses</td>
</tr>
<tr>
<td></td>
<td>It is possible to deposit BTC, ETH, LTC, NEO and BNB. Please check How to Register and Deposit on Binance²⁷ on how to query the corresponding addresses</td>
</tr>
<tr>
<td>Account Events</td>
<td>Supported</td>
</tr>
<tr>
<td>Account Balance</td>
<td>Supported.</td>
</tr>
<tr>
<td>Retrieval</td>
<td></td>
</tr>
</tbody>
</table>

### 23.30. Bitfinex

Bitfinex²⁸ is a cryptocurrency exchange. The Bitfinex adapter provides order execution, market data, reference data and account data functionality. Please see the Bitfinex API reference²⁹ page for technical details about the supported features.

The relevant properties for the Bitfinex adapter are defined inside the file conf-bfx.properties where they can be changed. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```json
"apiKey": "String", "label": "API Key"

bfx.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

"apiKey": "String", "label": "API Secret"

bfx.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

²⁷ https://www.binance.com/en/support/articles/115003764971
²⁸ https://www.bitfinex.com/
²⁹ https://docs.bitfinex.com/docs
A Bitfinex account is necessary in order to use Bitfinex adapter. Unique apiKey and apiSecret settings must be set to the actual values (either in the properties file or by setting a VM argument).

### 23.30.1. Bitfinex Order Constraints

AlgoTrader supports Bitfinex exchange and margin trading.

Market, limit and stop orders are supported. For exchange account trading, different order types are used (exchange market, exchange limit and exchange stop).

#### Table 23.7. Bitfinex constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>minimum order quantity</td>
</tr>
<tr>
<td>MaxQty</td>
<td>maximum order quantity</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>price precision</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>

Details: For details see: Bitfinex

Automatic order reconciliation (on WebSocket reconnect) is implemented for Bitfinex.

### 23.30.2. Bitfinex Account Management

#### Table 23.8. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>AlgoTrader uses the v1 API. Supported currencies can be found here. The withdrawal address and payment id have to be provided in the withdrawContext. Bitfinex supports 3 types</td>
</tr>
</tbody>
</table>

---

31. https://docs.bitfinex.com/v1/reference#rest-auth-withdrawal
### Functionality Supported Features

- **Supported Features**
  - Wallets: trading, exchange, and deposit. The default is exchange. It can be changed in `conf-bfx.properties` file.
  - **Deposit Addresses**
    - The available currencies can be found [here](https://api.bitfinex.com/v2/conf/pub:map:currency:label). Bitfinex supports three types of wallets: trading, exchange, and deposit. Note that the USDT deposit method only works for verified accounts, otherwise this method throws an exception.

### 23.31. Bitflyer

*Bitflyer*[^33] is a cryptocurrency exchange. The Bitflyer adapter supports order execution, market data, reference data, and account data functionality. Please see the [Bitflyer API reference](https://bitflyer.com/api) page for technical details.

#### Note

At this point (April 2018), Bitflyer does not yet support cross-border trading, so trading vs. USD is only possible with a US account.

The relevant properties for the Bitflyer adapter are defined inside the file `conf-bfl.properties` where they can be changed. Alternatively, the properties can be changed via [Section 2.4, "VM Options"](

```java
#
"type":"String","label":"API Key"
"bfl.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#
"type":"String","label":"API Secret"
"bfl.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#
"type":"String","label":"PubNub Subscribe Key"
"bfl.pubNubSubscribeKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#
"type":"Integer","label":"REST API Rate Limit Milliseconds"
"bfl.rateLimits = 40/1m

#
"type":"Boolean","label":"if true import all currencies, otherwise only those defined in ch.algotrader.enumeration.Currency"
"bfl.importAllPairs = true
```

[^33]: https://bitflyer.com/en-ja/
[^34]: https://bitflyer.com/api
A Bitflyer account is necessary in order to use Bitflyer adapter. Unique apiKey, apiSecret as well as market data subscription key settings must be set to the actual values (either in the properties file or by setting a VM argument).

23.31.1. Bitflyer Order Constraints

We support Bitflyer margin and exchange trading.

Bitflyer uses different instruments for exchange and margin trading (see Bitflyer margin trading.35)

- **Exchange**: BTCJPY with security.DESCRIPTION BTC/JPY@FLYR
- **Margin**: BTCJPY with security.DESCRIPTION BTC/JPY-FX@FLYR

Bitflyer supports market and limit orders.

**Table 23.9. BitFlyer constraints**

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>BTC/JPY: 0.001</td>
</tr>
<tr>
<td></td>
<td>ETH/BTC: 0.01</td>
</tr>
<tr>
<td></td>
<td>FX: BTC/JPY: 0.01</td>
</tr>
<tr>
<td>MaxQty</td>
<td>-</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>1 JPY - if Quote Currency is JPY</td>
</tr>
<tr>
<td></td>
<td>0.00001 BTC - if Quote Currency is BTC</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
<tr>
<td>Details</td>
<td>The BitFlyer limits are described in FAQ: BitFlyer amounts36</td>
</tr>
</tbody>
</table>

Automatic order reconciliation (on WebSocket reconnect) is implemented for Bitflyer.

---

35 https://lightning.bitflyer.com/About-Fx?lang=en
23.31.2. Bitflyer Account Management

Table 23.10. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Withdrawals</strong></td>
<td>It is possible to withdraw JPY for Japanese accounts, USD for U.S. accounts, and EUR for European accounts. The bank account id has to be provided in withdrawContext as <code>address</code> parameter.</td>
</tr>
<tr>
<td><strong>Deposit Addresses</strong></td>
<td>Bitflyer supports BTC, ETH, LTC, BCH, MONA, LSK. Before an address is returned, it needs to be enabled/created under the Account Funding section after logging in to the BitFlyer Web UI.</td>
</tr>
<tr>
<td><strong>Account Events</strong></td>
<td>Not supported</td>
</tr>
<tr>
<td><strong>Account Balance Retrieval</strong></td>
<td>Supported.</td>
</tr>
</tbody>
</table>

23.32. BitHumb Pro (Global)

**BitHumb Pro**[^37] is a cryptocurrency exchange. The BitHumb Pro adapter supports order execution, market data, reference data and account data functionality. Please see the [BitHumb API reference][^38] page for technical details.

The relevant properties for the BitHumb adapter are defined inside the file `conf-bhb.properties` where they can be changed. Alternatively the properties can be changed via [Section 2.4, "VM Options"](#):

```properties
# BitHumb adapter properties

#{"type":"String","label":"REST Url"}
bhb.restUrl = https://global-openapi.bithumb.pro/openapi/v1

#{"type":"String","label":"Web Socket Url"}
bhb.wssUrl = wss://global-api.bithumb.pro/message/realtime

#{"type":"String","label":"API Key"}
bhb.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
bhb.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"ch.algotrader.config.RateLimit","label":"REST API Rate Limit"}
bhb.rateLimits = 95/1s
```

[^37]: [https://www.bithumb.pro/en-us](https://www.bithumb.pro/en-us)
[^38]: [https://github.com/bithumb-pro/bithumb.pro-official-api-docs](https://github.com/bithumb-pro/bithumb.pro-official-api-docs)
A BitHumb account is necessary in order to use the BitHumb adapter. Unique apiKey, apiSecret as well as market data subscription key settings must be set to the actual values (either in the properties file or by setting a VM argument)

### 23.32.1. BitHumb Pro Order Constraints

BitHumb Pro doesn't support margin trading.

BitHumb Pro supports market and limit orders.

BitHumb Pro does not support order modifications.

Minimum and maximum price and quantity increments are security specific and can be looked up in the Security table after loading the reference data.

Automatic order reconciliation (on WebSocket reconnect) is implemented for BitHumb Pro.

Buy market orders sized in base currency are not supported by the exchange. An example is BTC-USDT, when buying or selling using a market order in it, the API expects the quantity to be specified in USDT, not in BTC. The adapter by default works around this by placing a buy limit order above the current best ask price in order to immediately execute the buy order. A disadvantage of this workaround is that account balance requirement on order placement is slightly higher than it would be with market order. Another option is to execute market orders with quantity defined in the secondary currency by converting the requested base currency amount using the current bid/ask prices. The disadvantage of this approach is the quantity executed by the exchange is subtracted with fees and the price might change during execution or on orders with quantity bigger that the relevant order book top level size.
Which makes it difficult to execute an exact amount defined in the primary currency. In case you want to configure this functionality, see configuration properties bhb.executeBuyMarketOrdersUsingLimitOrders and bhb.executeBuyMarketOrdersUsingLimitOrdersPercentage.

### 23.32.2. BitHumb Pro Account Management

#### Table 23.11. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Supported.</td>
</tr>
<tr>
<td>Deposit Addresses</td>
<td>Not supported. Deposit addresses are not available on the BitHumb Pro API at the moment.</td>
</tr>
<tr>
<td>Account Events</td>
<td>Not supported. Account events are not available on the BitHumb Pro API at the moment.</td>
</tr>
<tr>
<td>Account Balance</td>
<td>Supported.</td>
</tr>
<tr>
<td>Retrieval</td>
<td></td>
</tr>
</tbody>
</table>

### 23.33. BitMEX

*BitMEX*[^39] is a cryptocurrency futures exchange. The BitMEX adapter provides order execution, market data, reference data and account data functionality through REST and WebSocket API. Please see the *API reference*[^40] page for technical details about the supported features.

The relevant properties for the BitMEX adapter are defined inside the file `conf-bmx.properties` where they can be changed. Alternatively the properties can be changed via *Section 2.4, “VM Options”*:

```java
#{"type":"String","label":"API Key"}
bmx.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
bmx.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"Integer","label":"REST API Rate Limit Milliseconds"}
bmx.rateLimits = 60/1m

#{"type":"Integer","label":"API-level constant'}
bmx.balanceScale = 8
```

A BitMEX account is necessary in order to use the BitMEX adapter. Unique `apiKey` and `apiSecret` settings must be set to the actual values (either in the properties file or by setting a VM argument).

[^39]: https://www.bitmex.com/
[^40]: https://www.bitmex.com/app/apiOverview
Table 23.12. Supported Instruments

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Future</td>
<td>Regular future contracts expiring every 3 months. Please note that the XBT (BTC) contracts have a variable contract size. For more information please see Futures Guide.</td>
</tr>
<tr>
<td>PerpetualSwap</td>
<td>Bitcoin perpetual contract XBTUSD (BTCUSD) are represented by this class. For more information please see Perpetual Contract Specification.</td>
</tr>
<tr>
<td>Index</td>
<td>For a complete list of supported indices please see Indices.</td>
</tr>
</tbody>
</table>

BitMEX supports trading of the perpetual contract and of the futures.

The minimum quantity for all contracts is 1 contract (lot size = 1). Only integer number of contracts are allowed. The QUANTITY_SCALE for all securities is set to 0 and must not be changed.

Placing an order to buy one XBTUSD means buying the amount of Bitcoin worth 1 USD. For more information please consult the BitMEX perpetual contract details page.

23.33.1. BitMEX Order Constraints

BitMEX is a Futures exchange, so only margin trading is supported.

AlgoTrader support market, limit, stop and stop limit orders.

Table 23.13. BitMEX constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>1</td>
</tr>
<tr>
<td>MaxQty</td>
<td>-</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>1</td>
</tr>
<tr>
<td>MinPrice</td>
<td>1 Satoshi</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>1 Satoshi</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>

Automatic order reconciliation (on WebSocket reconnect) is implemented for BitMEX.

---

41 https://www.bitmex.com/app/futuresGuide
42 https://www.bitmex.com/app/contract/XBTUSD
43 https://www.bitmex.com/app/index/BXBT
44 https://www.bitmex.com/app/seriesGuide/XBT
23.33.2. BitMEX Account Management

Table 23.14. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Only BTC is supported</td>
</tr>
<tr>
<td>Deposit</td>
<td>Only BTC is supported</td>
</tr>
<tr>
<td>Addresses</td>
<td>Only BTC is supported</td>
</tr>
<tr>
<td>Account Events</td>
<td>Supported</td>
</tr>
<tr>
<td>Balance Retrieval</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

23.34. Bitstamp

*Bitstamp*[^45] is a cryptocurrency exchange. Please see the *API reference*[^46] page for the technical details.

Order and market data related functionality is provided via FIX/4.4 protocol. Account data and reference data is provided via REST API.

Bitstamp FIX/4.4 interface follows the standard closely, but offers only one session for both market data feed and trading operations. Bitstamp market data supports only limited number of cryptocurrency (Forex) securities. Order modifications are not supported. For more information about the Bitstamp FIX specification please have a look at the *Bitstamp public FIX interface*[^47].

The relevant properties for the Bitstamp adapter are defined inside the file `conf-bts.properties` where they can be changed. Alternatively the properties can be changed via *Section 2.4, “VM Options”*:

```properties
#{"type":"String","label":"API Key"}
bts.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
bts.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"Customer ID"}
bts.customerId = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"Integer","label":"REST API Rate Limit Milliseconds"}
bts.rateLimits = 40/1m

#{"type":"Boolean","label":"if true import all currencies, otherwise only those defined in ch.algotrader.enumeration.Currency"}
```

[^45]: https://www.bitstamp.net/
[^46]: https://www.bitstamp.net/api/
[^47]: https://www.bitstamp.net/fix/
A Bitstamp account is necessary in order to use Bitstamp adapter. Unique apiKey and apiSecret settings must be set to the actual values (either in the properties file or by setting a VM argument)

### 23.34.1. Bitstamp Order Constraints

Bitstamp does not support margin trading.

AlgoTrader supports market and limit orders.

At the time of writing this, Bitstamp does not support order modifications.

**Table 23.15. BitStamp constraints**

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MaxQty</td>
<td>-</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MinPrice</td>
<td>0.00001</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>0.00001</td>
</tr>
<tr>
<td>MinNotional</td>
<td>25 USD</td>
</tr>
</tbody>
</table>

Details: There is one additional rule for BitStamp: the order value should be at least USD 25. The USD 25 value is calculated by BitStamp with the latest market prices. This is however not validated on the AlgoTrader side. For further details see [BitStamp limits](https://www.bitstamp.net/article/upcoming-changes-fee-schedule/)

### 23.34.2. Bitstamp Account Management

**Table 23.16. Supported Functionality**

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Support for LTC, ETH, BTC, BCH, XRP. The withdrawal address has to be provided in the withdrawContext.</td>
</tr>
<tr>
<td>Deposit</td>
<td>Support for LTC, ETH, BTC, BCH, XRP</td>
</tr>
<tr>
<td>Account Events</td>
<td>Not supported</td>
</tr>
<tr>
<td>Account Balance</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

---

48 [https://www.bitstamp.net/article/upcoming-changes-fee-schedule/](https://www.bitstamp.net/article/upcoming-changes-fee-schedule/)
Order reconciliation is implemented by loading pending order statuses via Rest API when FIX session reconnects, similar to adapters that work via Rest/Websocket interfaces and not via FIX (see Section 23.5, "Automatic order reconciliation after re-connect"). The reason for this is that Bitstamp FIX API uses a single session for both market data and trading and the automatic order reconciliation using FIX Resend message doesn't work reliably. fix.cfg property ResetOnLogOn needs to be turned on otherwise the logon to the Bitstamp FIX API might fail.

### 23.35. Bittrex

Bittrex[^49] is a cryptocurrency exchange. Bittrex adapter allows trading spot instruments (e.g. BTCUSD or BTCUSDT).

Bittrex adapter supports order execution, market data, reference data and account data functionality. Please see the [Bittrex API reference][50] page for technical details.

The relevant properties of the Bittrex adapter are defined inside the file conf-btx.properties where they can be changed. Alternatively the properties can be changed via Section 2.4, "VM Options":

```java
#{"type":"String","label":"REST Url"}
btx.restUrl = https://api.bittrex.com/v3/

#{"type":"String","label":"Web Socket Url"}
btx.wssUrl = https://socket.bittrex.com/signalr

#{"type":"String","label":"API Key"}
btx.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
btx.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"BTX session qualifier"}
btx.sessionQualifier = BTX

#{"type":"ch.algotrader.config.RateLimit","label":"REST API Rate Limit"}
btx.rateLimits = 60/1m

#{"type":"Integer","label":"Number of maximum idle period between incoming messages in seconds"}
btx.webSocketTimeoutSeconds = 20

#{"type":"double","label":"Order book filter for filtering order book price levels"}
```

[^49]: https://global.bittrex.com/
[^50]: https://bittrex.github.io/api/v3
23.35.1. Bittrex Order Constraints

Bittrex exchange doesn’t support margin trading

Note
Bittrex adapter currently supports market and limit orders.

Bittrex adapter currently doesn’t support order modifications.

Time In Force values supported: Good-till-cancel (GTC), Fill-Or-Kill (FOK), PO (Post-Only), Immediate-Or-Cancel (IOC)

Bittrex adapter requires that either FOK or IOC TIF value is set with market orders. Market orders default their TIF to FOK instead of usual GTC.

Rest calls rate limit is set to 60 calls per 1 minute by default in btx.rateLimits configuration property.

23.35.1.1. Bittrex constraints

Order quantity increment is 0.00000001. Price increment and minimum quantity are security specific. They can be looked up in the Security table after running the reference data starter with Kraken Spot adapter configured: Chapter 20, Reference Data. For other trading restrictions applied by the exchange, see this article

23.35.2. Bittrex Account Management

Table 23.17. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Supported with all crypto-currencies.</td>
</tr>
<tr>
<td>Wallet History</td>
<td>Supported with all crypto-currencies. Loads history of deposits and withdrawals.</td>
</tr>
<tr>
<td>Deposit Addresses</td>
<td>Supported with all crypto-currencies.</td>
</tr>
<tr>
<td>Account Events</td>
<td>Account balance change events supported.</td>
</tr>
</tbody>
</table>

Blockfills adapter supports order execution and market data. Reference data is available in the provided sample data. Account data functionality is not available in Blockfills API.

Blockfills provides a fully functional UAT test environment and the adapter can be configured to use it.

Connection properties and credentials for the FIX adapter need to be set in `/bootstrap/conf/src/main/resources/fix.cfg`. You can copy there the content of the example `fix-bf.cfg` file at the same location.

Relevant properties of the Blockfills adapter can be found inside the file `conf-bf.properties` where they can be adjusted. Alternatively these properties can be changed via Section 2.4, “VM Options”:

```yaml
bf.orderbookfilter=1000
bf.supportedTifs=GTC,IOC,FOK,GTD
```

Reference data functionality is not available in the Blockfills FIX API. All available securities and other necessary values are available inside the sample data file `mysql-data.sql`.

### 23.36.1. Blockfills Order Constraints

Currently the following limitations and known issues exist:

- RFS and RFQ trading is not available via Blockfills FIX API at the moment.
- Leveraged trading is not available.
- Market, limit and stop orders are supported.
- Order modification is not supported.

[52] https://www.blockfills.com/
• Contract sizes, quantity and price increments are specific to different securities and are loaded from the Security table.

• Order reconciliation after reconnecting is provided automatically by the FIX protocol.

23.37. CoinAPI

CoinAPI\(^{53}\) is a market data gateway to multiple crypto exchanges. CoinAPI provides historical and live market data. It also provides reference data for the supported instruments, however it doesn't provide trading related functionality. Please see the API reference\(^{54}\) page for the technical details.

Historical data is available down to 1 second bars. Historical data availability varies by currency. Up to 100 daily requests can be placed for free. Consult their pricing\(^{55}\) if you require more.

Instruments and exchanges must have CNPID value setup in security and exchange database tables.

The relevant properties for the CoinAPI adapter are defined inside the file conf-cnp.properties where they can be changed. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```java
#{"type":"String","label":"API Key"}
cnp.apiKey = XXXXXXXXXXXXXXXXX

#{"type":"Integer","label":"REST API Rate Limit Milliseconds"}
cnp.rateLimits = 40/1m

#{"type":"Integer","label":"API-level constant'}
cnp.scale = 5

#{"type":"Boolean","label":"if true import all currencies, otherwise only those defined in ch.algotrader.enumeration.Currency"}
cnp.importAllPairs = true

#{"type":"String[]","label":"can contain values: trade, quote, book20"}
cnp.websocketUpdates = trade,quote
```

Unique apiKey and apiSecret settings must be set to the actual values (either in the properties file or by setting a VM argument)

23.38. Coinbase Pro

Coinbase\(^{56}\) is a cryptocurrency exchange. The Coinbase Pro adapter supports order execution, market data, reference data and account data functionality. Please see the Coinbase API reference\(^{57}\) page for technical details.

---

\(^{53}\) https://www.coinapi.io/
\(^{54}\) https://docs.coinapi.io/
\(^{55}\) https://www.coinapi.io/pricing
\(^{56}\) https://pro.coinbase.com
Note
Currently Coinbase Pro does not yet support cross-border trading, so trading vs. USD is only possible with a US account.

The relevant properties for the Coinbase Pro adapter are defined inside the file conf-cnb.properties where they can be changed. Alternatively the properties can be changed via Section 2.4, "VM Options":

```java
#{"type":"String","label":"Web Socket Url"}
cnb.webSocketUrl = wss://ws-feed.pro.coinbase.com
#cnb.webSocketUrl = wss://ws-feed-public.sandbox.pro.coinbase.com

#{"type":"Integer","label":"Number of maximum idle period between incoming messages in seconds"}
cnb.webSocketTimeoutSeconds = 10

#{"type":"String","label":"API Key"}
cnb.apiKey = XXX

#{"type":"String","label":"API Secret"}
cnb.apiSecret = XXX

#{"type":"String","label":"API passphrase"}
cnb.passphrase = XXX

#{"type":"String","label":"REST Url"}
cnb.restUrl = https://api.pro.coinbase.com
#cnb.restUrl = https://api-public.sandbox.pro.coinbase.com

#{"type":"Integer","label":"REST API Rate Limit Milliseconds"}
cnb.rateLimits = 5/1s

#{"type":"Boolean","label":"if true import all currencies, otherwise only those defined in ch.algotrader.enumeration.Currency"}
cnb.importAllPairs = true
```

A Coinbase Pro account is necessary in order to use Coinbase Pro adapter, but not needed for market data. Coinbase Pro provides a sandbox/testing environment, both for the public website (Sandbox website url\textsuperscript{58}) and the exchange. You'll need to generate (via the Coinbase website) and configure (see properties above) an API key and API secret in order to use the Coinbase Pro Adapter. The sandbox/testing environment has different URLs (see commented out values cnb.webSocketUrl, cnb.restUrl above).

\textsuperscript{57} https://docs.pro.coinbase.com
\textsuperscript{58} https://public.sandbox.pro.coinbase.com
23.38.1. Coinbase Pro Order Constraints

Coinbase Pro does not support margin trading.

AlgoTrader supports market, limit and stop limit orders.

**Note**

Coinbase Pro does not currently support order modifications.

### Table 23.18. Coinbase Pro constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>0.001 BTC</td>
</tr>
<tr>
<td></td>
<td>0.01 BCH</td>
</tr>
<tr>
<td></td>
<td>0.01 ETH</td>
</tr>
<tr>
<td></td>
<td>0.1 LTC</td>
</tr>
<tr>
<td>MaxQty</td>
<td>-</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>BaseMinSize scale</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>Quote Increment</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>

Automatic order reconciliation (on FIX reconnect) is implemented for Coinbase (required, as the Coinbase Pro FIX server does not support the `Resend` message).

23.38.2. Coinbase Pro Account Management

### Table 23.19. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Support for all cryptocurrencies</td>
</tr>
<tr>
<td>Deposit</td>
<td>Support for all cryptocurrencies. Note that the deposit address returned is for an account in Coinbase (non-PRO). The transfer to Coinbase Pro (system for trading) needs to be done manually via the Coinbase Pro website.</td>
</tr>
<tr>
<td>Addresses</td>
<td></td>
</tr>
</tbody>
</table>
23.39. CoinMarketCap

*CoinMarketCap*[^60] - Cryptocurrency Market Capitalizations is a website providing information about all existing crypto currencies and exchanges. The CoinMarketCap interface connects to the website via HTML and **REST API**[^61].

The CoinMarketCap interface provides the publicly available daily historical data and reference data for all listed crypto currencies. No account is necessary in order to use the CoinMarketCap adapter.

### Supported Features

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account Events</td>
<td>Not supported</td>
</tr>
<tr>
<td>Account Balance Retrieval</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

23.40. CryptoFinance

CryptoFinance is a cryptocurrency market maker/OTC trading services provider.

The CryptoFinance interface supports Market Data, Order Processing, Request for Quote (RFO) process as well as Reference Data.

The relevant properties for the CryptoFinance adapter are defined inside the file `conf-cf.properties` where they can be changed. Alternatively the properties can be changed via **Section 2.4, "VM Options"**:

```java
#{"type":"Long","label":"The id of the account to be used in conjunction with the tilde RFO process."}
cf.quoteAccountId = 209
#{"type":"String","label":"REST Url"}
cf.restUrl=https://broker-stage.cryptofinance.ch/api-oneshot/portal/v1
#{"type":"Boolean","label":"import all currencies or only those defined in ch.algotrader.enumeration.Currency"}
cf.importAllPairs=true
#{"type":"Boolean","label":"if true override existing security families fields: MIN_QTY, MAX_QTY, QTY_INCR, MIN_PRICE, MAX_PRICE, PRICE_INCR, MIN_NOTIONAL"}
cf.overrideSecurityFamilies=true
#{"type":"String","label":"Username for REST API"}
cf.username=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#{"type":"String","label":"Password for REST API"}
cf.password=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#{"type":"Integer","label":"REST API Rate Limit Milliseconds"}
cf.rateLimit=120/1m
#{"type":"String","label":"Price levels to subscribe per currency - comma separated key=value pairs"}
```

[^60]: https://coinmarketcap.com/
[^61]: https://coinmarketcap.com/api/
CryptoFinance does not have an order book but will stream prices to you at closest to requested level for each coin (e.g. CryptoFinance streams prices for 1, 5, 20 and 100 BTC, user requests price for 11 BTC, so price for 20BTC would be streamed to user.).

In order to populate the database with CryptoFinance Accounts, Exchanges, Security Families and Securities run the `ReferenceDataStarter with cFReferenceData` spring profile enabled and program argument: `all`. For further details please visit Chapter 20, Reference Data.

Note that time in force (TIF) for CryptoFinance Previously Indicated orders has to be Fill or Kill (FOK), and for non RFQ orders Good till Cancel (GTC) or Good till Day (GTD).

23.40.1. CryptoFinance Order Constraints

CryptoFinance supports previously indicated (for quote execution), market and limit orders. CryptoFinance also supports order cancelation.

Note

CryptoFinance does not currently support order modifications.

23.40.1.1. CryptoFinance Order Constraints

AlgoTrader supports CryptoFinance exchange trading.

Table 23.20. CryptoFinance constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>-</td>
</tr>
<tr>
<td>MaxQty</td>
<td>maximum order quantity</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>-</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>price precision</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>
23.41. Deribit

Deribit\(^{62}\) is a crypto exchange which provides trading on derivatives based on Bitcoin and Ethereum. The provided derivatives are futures, perpetual swaps and options. Derivatives are priced based on Bitcoin and Ethereum indexes that follow BTCUSD and ETHUSD price weighted from several exchanges.

Deribit adapter supports order execution, market data, reference data and account data functionality. All adapter functionality is implemented using FIX protocol except for Section 21.4, “Account Events” functionality (available if the Deribit account adapter is turned on) that use Websocket. Please see the Deribit API reference\(^{63}\) page for technical details and Deribit website\(^{64}\) for the list of available instruments and their historical prices.

Deribit provides fully functional test environment\(^{65}\) and Deribit adapter can be configured to use it.

Connection properties for FIX adapter need to be set in `/bootstrap/conf/src/main/resources/fix.cfg`. In case of Deribit adapter, `fix.cfg` file should not contain any connection credentials as they are loaded from properties. You can copy there the content of the example `fix-drb.cfg` file at the same location. Note that in order to use the test environment, the commented out `SocketConnectHost` and `SocketConnectPort` properties should be used instead of the ones that are uncommented:

```plaintext
# SocketConnectHost=testapp.deribit.com
# SocketConnectPort=9881
SocketConnectHost=www.deribit.com
SocketConnectPort=9880
```

Relevant properties of the Deribit adapter including connection credentials are defined inside the file `conf-drb.properties` where they can be adjusted. Alternatively these properties can be changed via Section 2.4, “VM Options”:

```json
{"type":"String","label":"Deribit WebSocket url"}
#drb.websocketUrl = wss://test.deribit.com/ws/api/v2/
drb.websocketUrl = wss://www.deribit.com/ws/api/v2/

{"type":"String","label":"API key."}
drb.apiKey=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

{"type":"String","label":"API secret."}
drb.apiSecret=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

{"type":"String","label":"Supported TIFs. Loaded by SupportedTifsConfig"}
drb.supportedTifs=FOK,IOC,GTC
```

\(^{62}\)https://www.deribit.com/
\(^{63}\)https://docs.deribit.com/v2/
\(^{64}\)https://www.deribit.com/main#/indexes
\(^{65}\)https://test.deribit.com/
Note that futures and perpetual swaps in Deribit are defined using contracts valued in the base currency instead of the quote currency. They are also referred to as inverse contracts instruments. An example is the BTC perpetual swap, with its contract valued in BTC. This is an equivalent of BTCUSD (XBTUSD) perpetual swap in BitMex, note that the instrument is not defined exactly the same way and its price may differ from the BitMex one.

Options are priced and settled in the crypto, BTC or ETH. For details see Deribit's documentation at [Options information](https://www.deribit.com/pages/docs/options).

Valuation of inverse contracts changes PnL calculation, but is handled by AlgoTrader automatically. For details see Deribit's documentation at [Perpetual Swaps information](https://www.deribit.com/pages/docs/perpetual) and [Futures information](https://www.deribit.com/pages/docs/futures).

Deribit index securities are loaded by the reference data adapter and market data is available on them. They are not tradable. Because Deribit only provides a single price for each index, the market data is delivered with bid and ask set both to this price and their respective quantities are set to 1.

### 23.41.1. Deribit Order Constraints

Currently the following limitations and known issues exist:

- All orders in Deribit are margined.
- Only market and limit orders are supported.
- Order modification is not supported.
- Contract sizes, quantity and price increments are specific to different securities and are loaded by the reference data starter into Security table.
- Deribit's FIX API uses the same session for market data and order placement and statuses. Resend messages from client to broker are not supported. Those are usually used for order reconciliation on restart.

---

66 [https://www.deribit.com/pages/docs/options](https://www.deribit.com/pages/docs/options)
67 [https://www.deribit.com/pages/docs/perpetual](https://www.deribit.com/pages/docs/perpetual)
68 [https://www.deribit.com/pages/docs/futures](https://www.deribit.com/pages/docs/futures)
or reconnect of FIX client which is AlgoTrader in this case. The order reconciliation is implemented on AlgoTrader side by requesting order statuses from Deribit using `OrderMassStatusRequest` message. This is triggered on `LOGGED_ON` session event (see Section 23.4, “Session life-cycle events”), which is triggered when AlgoTrader starts or the Deribit adapter is reconnected.

### 23.41.2. Deribit Account Management

**Table 23.21. Supported Functionality**

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Supported. Withdrawal address needs to be added to the account via the exchange website for security reasons.</td>
</tr>
<tr>
<td>Deposit Addresses</td>
<td>Supported.</td>
</tr>
<tr>
<td>Account Events</td>
<td>Supported.</td>
</tr>
<tr>
<td>Account Balance</td>
<td>Supported.</td>
</tr>
<tr>
<td>Retrieval</td>
<td></td>
</tr>
</tbody>
</table>

### 23.42. Enigma

Enigma is a cryptocurrency exchange that also supports RFQ process.

The Enigma interface supports Market Data, Order Processing, Request for Quote (RFQ) process as well as Reference Data.

The relevant properties for the Enigma adapter are defined inside the file `conf-egm.properties` where they can be changed. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```java
#{"type":"Long","label":"The id of the account to be used in conjunction with the Enigma RFQ process."}
egm.quoteAccountId=208
#{"type":"String","label":"REST Url"}
egm.restUrl=https://api.enigma-securities.io
#{"type":"String","label":"Infrastructure type of Enigma REST API (one of: prod, dev)"}
egm.restInfra=prod
#{"type":"String","label":"Enigma username to use with REST/socket connection"}
egm.username=XXX
#{"type":"String","label":"Enigma password to use with REST/socket connection"}
egm.password=XXX
#{"type":"int","label":"Interval (in seconds) between REST authentication refreshes"}
egm.authRefreshIntervalSeconds=3600
#{"type":"ch.algotrader.config.RateLimit","label":"REST API Rate Limit Milliseconds"}
egm.rateLimit=1/1s
```
Enigma does not have an order book but will stream prices to you at requested level for each coin (e.g. 5BTC).

In order to populate the database with Enigma Accounts, Exchanges, Security Families and Securities run the ReferenceDataStarter with eGMReferenceData spring profile enabled and program argument: all. For further details please visit Chapter 20, Reference Data.

Note that time in force (TIF) for Enigma for non RFQ orders must be Fill or Kill (FOK), Immediate or Cancel (IOC), Good till Cancel (GTC) or Good till Day (GTD).

Note that Enigma allows RFQ process not for all instruments. Only instruments that are provided with egmId can be quoted. Also, production or test environment can be choosed to execute rfq (egm.restInfra parameter).

### 23.42.1. Enigma Order Constraints

Enigma supports previously indicated (for quote execution), market, limit, stop and stop limit orders. CryptoFinance also supports order modifications and cancelation.

#### 23.42.1.1. Enigma Order Constraints

AlgoTrader supports Enigma exchange trading.

### Table 23.22. Enigma constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>minimum order quantity or 0.01 if not provided</td>
</tr>
<tr>
<td>MaxQty</td>
<td>maximum order quantity or non if not provided</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>quantity precision</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>price precision</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>
23.43. eToroX

eToroX\(^{69}\) is a cryptocurrency exchange. The eToroX adapter provides order execution, market data, reference data and account functionality. Please see the eToroX API reference\(^{70}\) page for technical details about the supported features.

The relevant properties for the eToroX adapter are defined inside the file `conf-etx.properties` where they can be changed. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```json
"type":"String","label":"REST Url"
etx.apiHost=https://xxxxxxxxxxx.hft.etorox.com
"type":"String","label":"Web Socket Url"
etx.websocketHost=https://xxxxxxxxxxx.hft-ws.etorox.com
"type":"String","label":"API Key / public key generated by eToroX"
etx.apiKey=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
"type":"String","label":"Path to API Secret file where private key generated by eToroX is stored. The key provided by EtoroX webpage needs to be converted first: https://gist.github.com/etorox/094eb6d22cf83d72218d2d28ff304bc4"
etx.apiSecretPath=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
"type":"Boolean","label":"if true import all currencies, otherwise only those defined in ch.algotrader.enumeration.Currency"
etx.importAllPairs=true
```

An eToroX account is necessary in order to use eToroX adapter. Unique `apiHost`, `websocketHost`, `apiKey` and `apiSecret` settings must be set to the actual values (either in the properties file or by setting a VM argument).

Note

API host, websocket host, API key and API secret are uniquely generated for eToroX accounts. API secret provided by eToroX platform should be converted as described \(^{71}\). Then the secret should be stored in a file and filesystem path to it should be set under `etx.apiSecretPath`.

23.43.1. eToroX Order Constraints

AlgoTrader supports eToroX exchange trading.

Only limit orders are supported via the API

Table 23.23. eToroX constraints

<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinQty</td>
<td>minimum order quantity</td>
</tr>
</tbody>
</table>

\(^{69}\) [https://www.etorox.com/](https://www.etorox.com/)

\(^{70}\) [https://etorox.github.io/docs/#/](https://etorox.github.io/docs/#/)

\(^{71}\) [https://gist.github.com/etorox/094eb6d22cf83d72218d2d28ff304bc4/](https://gist.github.com/etorox/094eb6d22cf83d72218d2d28ff304bc4/)
<table>
<thead>
<tr>
<th>Constraint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxQty</td>
<td>maximum order quantity</td>
</tr>
<tr>
<td>QtyIncr</td>
<td>0.00000001</td>
</tr>
<tr>
<td>MinPrice</td>
<td>-</td>
</tr>
<tr>
<td>MaxPrice</td>
<td>-</td>
</tr>
<tr>
<td>PriceIncr</td>
<td>price precision</td>
</tr>
<tr>
<td>MinNotional</td>
<td>-</td>
</tr>
</tbody>
</table>

Automatic order reconciliation (on WebSocket reconnect) is implemented for eToroX.

## 23.43.2. eToroX Account Management

### Table 23.24. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Supported</td>
</tr>
<tr>
<td>Deposit</td>
<td>Supported</td>
</tr>
<tr>
<td>Addresses</td>
<td>Not supported by the API</td>
</tr>
<tr>
<td>Account Events</td>
<td></td>
</tr>
<tr>
<td>Account Balance</td>
<td>Supported</td>
</tr>
<tr>
<td>Retrieval</td>
<td></td>
</tr>
</tbody>
</table>

## 23.44. Gemini

*Gemini*[^72] is a crypto-currency exchange providing trading on spot crypto pairs.

Gemini adapter supports trading, market data and account data functionality. Reference data is not available on Gemini’s APIs and is implemented in AlgoTrader’s sample data. Trading and market data functionality is implemented using Gemini’s FIX API. Account data functionality using Gemini’s Rest API. Please see the *Gemini API reference*[^73] page for technical details.

Gemini provides *Sandbox environment*[^74] replicating the full exchange functionality (market data is restricted to BTCUSD).

The use of FIX API by the adapter allows requesting low latency cross-connect with Gemini. Note that FIX API access requires an on-boarding procedure with Gemini, this includes FIX API access to their Sandbox environment. For details, please see Gemini’s *documentation*[^75].

[^72]: https://gemini.com/
[^73]: https://docs.gemini.com/
[^74]: https://exchange.sandbox.gemini.com
[^75]: https://docs.gemini.com/fix-api/#download
Connection properties for the FIX adapter need to be set in /bootstrap/conf/src/main/resources/fix.cfg. You can copy there the content of the example fix-gemini.cfg file at the same location.

```
[default]
ConnectionType=initiator
HeartBtInt=30
ReconnectInterval=5
FileStorePath=files/fix
FileLogPath=log
FileLogHeartbeats=N
FileIncludeMilliseconds=Y
FileIncludeTimeStampForMessages=Y
SLF4JLogHeartbeats=Y
ResetSeqNumFlag=N

[session]
SessionQualifier=GMN-T
BeginString=FIX.4.4
# UAT
SocketConnectHost=8.44.203.78
SocketConnectPort=1538
TargetCompID=???
SenderCompID=???
SocketUseSSL=N
TimeZone=America/New_York
StartTime=17:00:00
EndTime=17:00:00
Inactive=Y
ResetOnLogon=N
DataDictionary=gmn/gemini-fix-dictionary.xml
UseDataDictionary=Y
# pending orders are by default canceled by Gemini exchange after the client disconnects
CancelOnDisconnect=N

[session]
SessionQualifier=GMN-MD
BeginString=FIX.4.4
# UAT
SocketConnectHost=8.44.203.78
SocketConnectPort=1539
TargetCompID=???
SenderCompID=???
SocketUseSSL=N
TimeZone=America/New_York
StartTime=17:00:00
EndTime=17:00:00
Inactive=Y
```
ResetOnLogon=Y  
DataDictionary=gmn/gemini-fix-dictionary.xml  
UseDataDictionary=Y

Relevant properties of the Gemini adapter including connection credentials are defined inside the file `conf-gmn.properties` where they can be adjusted. Alternatively these properties can be changed via Section 2.4, "VM Options".

```java
# Gemini adapter configuration. FIX connection configuration is in fix.cfg.

#{"type":"String","label":"Rest API URL"}
gmn.restUrl = https://api.gemini.com
# gmn.restUrl = https://api.sandbox.gemini.com

#{"type":"ch.algotrader.config.RateLimit","label":"REST API Rate Limit"}
gmn.rateLimits = 5/1s

#{"type":"String","label":"API Key"}
gmn.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
gmn.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"double","label":"Order book filter for filtering order book price levels. Used by generic order book handling mechanism.s"}
gmn.orderbookfilter=1000

#{"type":"Boolean","label":"If true, buy market orders will be executed by placing a limit order significantly below current market price. If false, the regular market buy order is used and the notional quantity value is calculated from the current ask price."}
gmn.executeBuyMarketOrdersUsingLimitOrders = true

#{"type":"Double","label":"Distance from current market price where limit orders are executed when executeBuyMarketOrdersUsingLimitOrders. e.g. 0.02=2%"}
gmn.executeBuyMarketOrdersUsingLimitOrdersPercentage = 0.02

#{"type":"String","label":"Supported TIFs. Loaded by TifsConfig"}
gmn.supportedTifs=GTC,IOC,FOK,ATC,PO
```

Order reconciliation works automatically using standard FIX API messages. Note that Gemini verify there is no gap in sequence numbers on login. Under some special circumstances, e.g. network timeouts, the latest sent sequence number on the AlgoTrader side may become outdated. In case the trading adapter is unable to connect, you may try changing the `fix.cfg` setting `ResetOnLogon` to value `Y` and restart AlgoTrader. Note
that this setting in effect also disables order reconciliation and should be changed to the default value of N afterwards.

23.44.1. Gemini Order Constraints

Currently the following limitations and known issues exist:

- Leveraged (margin) orders are not supported by Gemini.
- Market, limit and stop limit orders are supported.
- Order modification is not supported.
- Contract sizes, quantity and price increments are specific to different securities and are set in sample data (mysql-data.sql) in the Security table.
- Subscribed market data can not be unsubscribed in the FIX API. In case the unsubscribe is requested, the Gemini adapter stops creating market data related events, but the physical delivery of market data from the exchange to the adapter continues.
- Pending orders are by default automatically cancelled by the exchange after the client disconnects. This functionality can be turned off by setting CancelOnDisconnect=N in trading session settings in fix.cfg.

Buy market orders sized in base currency are not supported by the exchange. An example is BTCUSD, when buying or selling using a market order in it, the API expects the quantity to be specified in USD, not in BTC. The adapter by default works around this by placing a buy limit order above the current best ask price in order to immediately execute the buy order in specified BTC quantity. A disadvantage of this workaround is that account balance requirement on order placement is slightly higher than it would be with market order. Another option is to execute market orders with quantity defined in the secondary currency by converting the requested base currency amount using the current best ask price. The disadvantage of this approach is the quantity executed by the exchange may be slightly smaller than originally requested due to rounding, move of the price or when the order quantity crosses more levels in the sell part of the order book. In case a smaller amount is executed, the adapter still creates an EXECUTED (full fill) order status. In case you want to configure this functionality, see configuration properties gmn.executeBuyMarketOrdersUsingLimitOrders and gmn.executeBuyMarketOrdersUsingLimitOrdersPercentage.

23.44.2. Gemini Account Management

Table 23.25. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Supported</td>
</tr>
<tr>
<td>Deposit</td>
<td>Supported</td>
</tr>
<tr>
<td>Addresses</td>
<td></td>
</tr>
<tr>
<td>Account Events</td>
<td>Not supported by the API</td>
</tr>
</tbody>
</table>


### Functionality Supported Features

<table>
<thead>
<tr>
<th>Account Balance Retrieval</th>
</tr>
</thead>
</table>

23.45. Huobi Spot

*Huobi* is a crypto exchange which provides trading on spot coin based instruments (e.g. BTCUSDT) and derivatives. Huobi Spot adapter provides access to spot instruments (see the [website](https://www.huobi.com/en-us/)).

Huobi Spot adapter supports order execution, market data, reference data and account data functionality. All adapter functionality is implemented using the exchange's Rest and Websocket APIs. Please see the Huobi Spot API reference ([link](https://www.huobi.com/en-us/exchange/)) page for technical details.

Relevant properties of the Huobi Spot adapter are defined inside the file `conf-hbi.properties` where they can be adjusted. Alternatively these properties can be changed via [Section 2.4, "VM Options"](https://huobiapi.github.io/docs/spot/v1/en/#introduction):

```properties
# note that urls below can use their non-AWS versions when not running on AWS.
#
#{"type":"String","label":"REST Url"}
# hbi.restUrl = https://api-aws.huobi.pro
hbi.restUrl = https://api.huobi.pro
#
#{"type":"String","label":"Market data WebSocket Url"}
# hbi.marketDataWebsocketUrl  = wss://api-aws.huobi.pro/ws
hbi.marketDataWebsocketUrl  = wss://api.huobi.pro/ws
#
#{"type":"String","label":"Account and order WebSocket Url"}
# hbi.accountAndOrderWebsocketUrl  = wss://api-aws.huobi.pro/ws/v1
hbi.accountAndOrderWebsocketUrl  = wss://api.huobi.pro/ws/v1
#
#{"type":"String","label":"API Key"}
# hbi.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
hbi.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#{"type":"String","label":"API Secret"}
# hbi.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
hbi.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#{"type":"ch.algotrader.config.RateLimit","label":"REST API Rate Limit"}
# hbi.rateLimits = 10/1s
hbi.rateLimits = 10/1s
#
#{"type":"Integer","label":"Number of maximum idle period between incoming messages in seconds"}
# hbi.websocketTimeoutSeconds = 20
hbi.websocketTimeoutSeconds = 20
```

---

*76* [https://www.huobi.com/en-us/]

*77* [https://www.huobi.com/en-us/exchange/]

*78* [https://huobiapi.github.io/docs/spot/v1/en/#introduction]
Buy market orders sized in base currency are not supported by the exchange. An example is BTCUSDT, when buying or selling using a market order in it, the API expects the quantity to be specified in USDT, not in BTC. The adapter by default works around this by placing a buy limit order above the current best ask price in order to immediately execute the buy order. A disadvantage of this workaround is that account balance requirement on order placement is slightly higher than it would be with market order. Another option is to execute market orders with quantity defined in the secondary currency by converting the requested base currency amount using the current bid/ask prices. The disadvantage of this approach is the quantity executed by the exchange is subtracted with fees and the price might change during execution or on orders with quantity bigger that the relevant order book top level size. Which makes it difficult to execute an exact amount defined in the primary currency. In case you want to configure this functionality, see configuration properties `hbi.executeBuyMarketOrdersUsingLimitOrders` and `hbi.executeBuyMarketOrdersUsingLimitOrdersPercentage`.

## 23.45.1. Huobi Spot constraints

Order modification is not supported by Huobi.

Quantities, increments and other constraints are security specific. They can be looked up in the Security table after running the reference data starter with Huobi Spot adapter configured: Chapter 20, Reference Data.

## 23.45.2. Huobi Spot Account Management

### Table 23.26. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Supported with all crypto-currencies with exceptions, for details see the <a href="https://www.huobi.com/en-us/finance/">Huobi webpage</a>.</td>
</tr>
<tr>
<td>Wallet History</td>
<td>Supported with all crypto-currencies with exceptions, for details see the <a href="https://www.huobi.com/en-us/finance/">Huobi webpage</a>.</td>
</tr>
</tbody>
</table>

Automatic order reconciliation (on market data WebSocket reconnect) is implemented for Huobi Spot.

23.46. Kraken Spot

Kraken is a cryptocurrency exchange. Kraken Spot adapter allows trading spot instruments (e.g. BTCUSD), and supports order execution, market data, reference data and account data functionality. Please see the Kraken API reference page for technical details.

The relevant properties of the Kraken Spot adapter are defined inside the file conf-kks.properties where they can be changed. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```java
#{"type":"String","label":"REST Url"}
kks.restUrl = https://api.kraken.com

#{"type":"String","label":"Secured WebSocket Url"}
kks.securedWebSocketUrl = wss://ws-auth.kraken.com
#{"type":"String","label":"Unsecured WebSocket Url"}
kks.unsecuredWebSocketUrl = wss://ws.kraken.com

#{"type":"String","label":"API Key"}
kks.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
kks.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"ch.algotrader.config.RateLimit","label":"REST API Rate Limit"}
kks.rateLimits = 60/1m

#{"type":"Boolean","label":"if true override existing security families fields: MIN_QTY, MAX_QTY, QTY_INCR, MIN_PRICE, MAX_PRICE, PRICE_INCR, MIN_NOTIONAL"}
kks.overrideSecurityFamilies = true
```

---

81 https://www.huobi.com/en-us/finance/
82 https://www.kraken.com/
83 https://www.kraken.com/features/api
23.46.1. Kraken Spot Order Constraints

Kraken Spot exchange supports margin trading at leverage levels defined per security. Each margin order can have one of several leverage options specified. Availability of leverage level options is different between different securities and is not available with some securities. Margin orders from AlgoTrader are set to use leverage of size 2 by default. This value can be changed using `kks.leverageToUseWithMarginOrders` configuration option. Individual orders can override the default setting by having `leverage` integer order property specified. Note that if you submit an order with a leverage not available with the security in Kraken, your order will be rejected by the exchange. The availability of leverage options can be verified using [Kraken website](https://trade.kraken.com).

Note

Kraken Spot adapter currently supports market, limit and stop orders. Stop Limit orders have been implemented but are currently not available in Kraken.

Kraken Adapter currently doesn't support order modifications.

Note that pending orders placed from AlgoTrader only appear at Kraken website after you fully refresh it.

German residents (users with German address) need to agree with Kraken's [trading agreement](https://support.kraken.com/hc/en-us/articles/360036157952) for legal reasons. Confirming the agreement is done on AlgoTrader side by setting the configuration property `kks.tradingAgreementApproved` to true. It defaults to false.

Time In Force values supported: Good-till-cancel (GTC), PO (Post-Only), Day Order (DAY), Good-till-Day (GTD)

---

84 [https://trade.kraken.com](https://trade.kraken.com)
85 [https://support.kraken.com/hc/en-us/articles/360036157952]
Rest calls rate limits in Kraken have complicated rules and it is recommended by Kraken to make just 1 order placement per second in general. The overall rate limit is set to 60 calls per 1 minute by default in kks.rateLimits configuration property.

23.46.1.1. Kraken Spot constraints

Quantities, increments and other constraints are security specific. They can be looked up in the Security table after running the reference data starter with Kraken Spot adapter configured: *Chapter 20, Reference Data*.

23.46.2. Kraken Spot Account Management

**Table 23.27. Supported Functionality**

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Supported with all crypto-currencies.</td>
</tr>
<tr>
<td>Wallet History</td>
<td>Supported with all crypto-currencies. Loads history of deposits and withdrawals.</td>
</tr>
<tr>
<td>Deposit Address</td>
<td>Supported with the following currencies: Bitcoin, Litecoin, Ether (Hex), Zcash (Transparent), ADA, Monero, Ripple XRP, Stellar XLM, Bitcoin Cash, REP, SynapsePay (US Wire), Dogecoin, MLN, GNO, QTUM, XTZ, Cosmos, EOS, Dash.</td>
</tr>
<tr>
<td>Account Events</td>
<td>Not supported.</td>
</tr>
<tr>
<td>Account Balance</td>
<td>Supported.</td>
</tr>
<tr>
<td>Retrieval</td>
<td></td>
</tr>
</tbody>
</table>

Automatic order reconciliation (on WebSocket reconnect) is implemented for Kraken Spot.

23.47. LMAX Digital

Supports trading of some of the most known digital currencies.

LMAX Digital implementation of the FIX/4.4 protocol has some peculiarities:

- Uses custom message dictionary
- Supports DAY, GTC, IOC and FOK time-in-force parameters for market and limit orders.
- Supports only DAY and GTC time-in-force parameters for stop orders (stop and stop-limit).
- supports trading status signaling temporary suspension and resumption in trading of individual securities.

23.47.1. LMAX Digital Order Constraints

There are several known constraints and limitations:

- Margin trading is not supported.
• Market and limit orders: **DAY, GTC, IOC and FOK** time-in-force supported.

• Stop and stop-limit orders: **DAY and GTC** time-in-force supported.

• Order modification (Cancel&Replace) supported only for limit orders.

• Order modification changes the time-in-force from its original value to **DAY**.

### 23.48. OKEx and OKCoin

**OKEx**[^okex] and **OKCoin**[^okcoin] are related cryptocurrency exchanges. The difference between them is that OKCoin allows trading fiat currency based instruments (e.g. BTCUSD) while OKEx does not. OKEx allows trading derivatives (futures, perpetual swaps and options).

OKEx and OKCoin adapters supports order execution, market data, reference data and account data functionality. Please see the [OKEx API reference][okex_api] and [OKCoin API reference][okcoin_api] pages for technical details.

The properties of the OKEx and OKCoin adapters are defined inside the files `conf-okex.properties` and `conf-okcoin.properties`. Alternatively the properties can be changed via Section 2.4, “VM Options” For OKCoin the properties are the same as below with suffix `okcoin`:

```properties
#{"type":"String","label":"OKEx REST endpoint url"}
okex.restUrl = https://www.okex.com

#{"type":"String","label":"OKEx websocket url"}
okex.wssUrl = wss://real.okex.com:8443/ws/v3

#{"type":"String","label":"API Key"}
okex.apiKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API Secret"}
okex.apiSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"API passphrase"}
okex.passPhrase = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"String","label":"Fund password for withdrawals etc."}
okex.fundPassword = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#{"type":"Double","label":"Network transaction fee on withdrawals is set as this distance between minimum and maximum OKEx recommended value. Zero uses minimum recommended, 1 maximum recommended fee."}
okex.recommendedNetworkTransactionFeeRatio = 0.05
```

[^okex]: https://www.okex.com/
[^okcoin]: https://www.okcoin.com/
[^okex_api]: https://www.okex.com/docs/en/
[^okcoin_api]: https://www.okcoin.com/docs/en/
Note that some futures and perpetual swaps in OKEx are defined using contracts valued in the base currency instead of the quote currency. They are also referred to as inverse contracts instruments. An example is the perpetual swap BTCUSD, with its contract valued in BTC. This is an equivalent of BTCUSD (XBTUSD) perpetual swap in BitMex, note that the instrument is not defined exactly the same way and its price may differ from the BitMex one. Inverse contract instruments are called 'Coin Margined' on the OKEx website. Regular non-inverse contract instruments are 'USDT margined' swaps/futures.

Valuation of inverse contracts changes PnL calculation, but is handled by AlgoTrader automatically. For details see OKEx's documentation at [Perpetual Swaps information](https://www.okex.com/en/futureTrade/beforePerpetualFuture) and [Futures information](https://www.okex.com/en/futureTrade/beforeFutures).
As of January 2020, OKEx provides options priced based on their BTC-USD index. All options are priced and settled in BTC.

23.48.1. OKEx/OKCoin Order Constraints

OKEx and OKCoin exchanges support margin trading at leverage levels defined per security. For leverage sizes, see [OKEx margin instruments page](https://www.okex.com/en/futureTrade/beforePerpetualFuture) and [OKCoin margin instruments page](https://www.okcoin.com/spot/trade?type=2).

Note

OKEx and OKCoin support market and limit orders only.

OKEx and OKCoin do not support order modifications.

Buy market orders sized in base currency are not supported for non-derivative instruments by the exchange. An example is BTCUSDT, when buying or selling using a market order in it, the API expects the quantity to be specified in USDT, not in BTC. The adapter by default works around this by placing a buy limit order above the current best ask price in order to immediately execute the buy order. A disadvantage of this workaround is that account balance requirement on order placement is slightly higher than it would be with market order. Another option is to execute market orders with quantity defined in the secondary currency by converting the requested base currency amount using the current bid/ask prices. The disadvantage of this approach is the quantity executed by the exchange is subtracted with fees and the price might change during execution or on orders with quantity bigger that the relevant order book top level size. Which makes it difficult to execute an exact amount defined in the primary currency. In case you want to configure this functionality, see configuration properties [okex.executeBuyMarketOrdersUsingLimitOrders](https://github.com/AlgoTrader/AlgoTrader#okex.executebuymarketordersusinglimitorders) and [okex.executeBuyMarketOrdersUsingLimitOrdersPercentage](https://github.com/AlgoTrader/AlgoTrader#okex.executebuymarketordersusinglimitorderspercentage).

Non-derivative instruments’ Websocket status changes messages do not include fees that were applied to fills. These are by default loaded by AlgoTrader using a Rest call after a message with a full fill is received. This functionality can be turned off using the configuration property [okex.loadFilledOrderFeesViaRestIfNecessary](https://github.com/AlgoTrader/AlgoTrader#okex.loadfilledorderfeesviarestifnecessary).

Quantities, increments and other constraints are security specific. They can be looked up in the Security table after running the reference data starters with OKEx and OKCoin adapters configured: Chapter 20, Reference Data.

23.48.2. OKEx/OKCoin Account Management

Table 23.28. Supported Functionality

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Supported Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdrawals</td>
<td>Supported with all crypto-currencies. Withdrawals require the output address to exist in the address book. Withdrawals can be only executed with your OKEx and OKCoin funding</td>
</tr>
</tbody>
</table>

92 https://www.okex.com/spot/marginTrade
93 https://www.okcoin.com/spot/trade?type=2

323
### Functionality Supported Features

**Wallet History**
- Supported with all crypto-currencies. Loads history of deposits and withdrawals in the funding account.

**Deposit Addresses**
- Supported with all crypto-currencies

**Account Events**
- Supported. Account events deliver balance on a non-funding account after a transfer. Transfers to and from the funding account with external accounts are not supported by the API and the adapter. If the transfer is executed between 2 non-funding accounts, 2 account events are created, one for each.

**Account Balance Retrieval**
- Supported.

Automatic order reconciliation (on WebSocket reconnect) is implemented for OKEx and OKCoin.
Chapter 24.

Execution Algos

24.1. Existing Execution Algos

AlgoTrader provides several built-in Execution Algos.

SlicingOrder

The Slicing Algo is only recommended for traditional assets like Equities and derivatives. It is disabled for cryptocurrencies, consider the TWAP or VWAP algs for similar functionality which will be compatible with assets traded on crypto exchanges.

Splits an order into several child orders. child order quantities and time in the market are randomized. The SlicingOrder has the following order properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Unit</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minQuantity</td>
<td>Minimum quantity for each child order</td>
<td>BigDecimal</td>
<td></td>
</tr>
<tr>
<td>maxQuantity</td>
<td>Maximum quantity for each child order</td>
<td>BigDecimal</td>
<td></td>
</tr>
<tr>
<td>minVolPct</td>
<td>Minimum % of volBid / volAsk to take</td>
<td>double</td>
<td>0%</td>
</tr>
<tr>
<td>maxVolPct</td>
<td>Maximum % of volBid / volAsk to take</td>
<td>double</td>
<td>100%</td>
</tr>
<tr>
<td>minDuration</td>
<td>Minimum duration of each child order</td>
<td>seconds</td>
<td>1.0</td>
</tr>
<tr>
<td>maxDuration</td>
<td>Maximum duration of each child order</td>
<td>seconds</td>
<td>1.0</td>
</tr>
<tr>
<td>minDelay</td>
<td>Minimum delay between two child orders</td>
<td>seconds</td>
<td>1.0</td>
</tr>
<tr>
<td>maxDelay</td>
<td>Maximum delay between two child orders</td>
<td>seconds</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The quantity of each child order is randomized between minVolPct and maxVolPct of the current volume offered at the exchange. In addition minQuantity and maxQuantity restriction can be imposed. If maxVolPct is zero or 100%, then the current market volume will not be considered when sizing the order. If maxQuantity is zero, then no maximum quantity will be enforced on top of the market volume restriction. The SlicingOrder will make sure that the remainingQty for the next child order is greater than minQuantity. Maximum quantity rules have precedence over minimum quantity rules.

Example:

minVolPct: 25%, minQuantity: 20, maxVolPct: 90%, maxQuantity: 100, BUY order, quantity: 40, vol ask: 10

minimum quantity: Max(25% x 10, 20) = 20

maximum quantity: Min(90% x 10, 100) = 9
This will result in an order of quantity 9

Each order will stay in the market for \( \text{minDuration} \) to \( \text{maxDuration} \) seconds (if it is not filled before that). Between each child order there will be a random delay of \( \text{minDelay} \) to \( \text{maxDelay} \) seconds. In addition, the SlicingOrder has a sophisticated pricing logic. For a BUY order the first child order will be place 1 tick below the Ask. For a SELL order the first tick will be placed one tick above the Bid. Depending on whether a child order gets filled, the price of the next child order is adjusted. If a child order gets filled, the price of the next child order will be reduced by one tick (for BUY orders) but it will always be at least one tick above the Bid. If the child order does not get filled, the price of the next child order is increased by one tick (for BUY orders) but it will never be higher than the ask. A SlicingOrder can be created and sent as follows:

```java
SlicingOrder order = new SlicingOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setMinQuantity(BigDecimal.valueOf(10));
order.setMaxQuantity(BigDecimal.valueOf(100));
order.setMinVolPct(0.01);
order.setMaxVolPct(0.1);
order.setMinDuration(1);
order.setMaxDuration(5);
order.setMinDelay(1);
order.setMaxDelay(5);
getOrderService().sendOrder(order);
```

Alternatively Section 17.2.1, “Order Preferences” can be used to create a SlicingOrder. The AlgoTrader sample data contains an OrderPreference named SLICING (with the default values shown in the table above) which allows placing a SlicingOrder as follows:

```java
Order order = getOrderService().createOrderByOrderPreference("SLICING");
order.setStrategy(strategy);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
getOrderService().sendOrder(order);
```

VWAPOrder

The VWAPorder seeks to achieve the Volume-Weighted Average price (VWAP)\(^1\). VWAP is a trading benchmark used by many institutional investors. VWAP is calculated by adding up the market value

---

\(^1\) https://en.wikipedia.org/wiki/Volume-weighted_average_price
traded for every transaction (price multiplied by number of contracts traded) and then dividing by the total contract traded. The VWAPOrder is based on the AdaptiveOrder (see) below and uses its pricing logic. The VWAPOrder has the following order properties in addition to the ones defined by the AdaptiveOrder.

Table 24.2. VWAPOrder

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Unit</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lookbackPeriod</td>
<td>look back period</td>
<td>days</td>
<td>5</td>
</tr>
<tr>
<td>bucketSize</td>
<td>size of each historical volume bucket</td>
<td>Duration</td>
<td>MIN_15</td>
</tr>
</tbody>
</table>

The VWAPOrder retrieves historical prices for the number of days specified in the `lookbackPeriod` parameter and splits the trading day into buckets with a length in minutes according to the `bucketSize` parameter.

When a VWAPOrder is either fully-executed or cancelled a message containing the average price, the benchmark price as well as the execution duration and number of executions is logged to the console. For the VWAPOrder to work a historical data adapter will need to be enabled, see Section 7.2.7, “Historical Data Service”.

A VWAPOrder can be created and sent as follows:

```java
VWAPOrder order = new VWAPOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setBucketSize(Duration.MIN_10);
order.setLookbackPeriod(10);
order.setDuration(600)
order.setSliceLength(10)
order.setCancelTime(0.5)
order.setTimeRand(0.25)
order.setQtyRand(0.25)
order.setIncrement(0.05)
order.setInitialOffset(0.8)
order.setMinOffset(0.05)
order.setMaxOffset(1.0)

getOrderService().sendOrder(order);
```

Alternatively Section 17.2.1, “Order Preferences” can be used to create a VWAPOrder. The AlgoTrader sample data contains an OrderPreference named VWAP (with the default values shown in the table above) which allows placing a VWAPOrder as follows:
Order order = getOrderService().createOrderByOrderPreference("VWAP");
order.setStrategy(strategy);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);

getOrderService().sendOrder(order);

TWAPOrder

The TWAPOrder seeks to achieve the **Time-Weighted Average price (TWAP)**. TWAP is a trading benchmark used by many institutional investors. TWAP is derived by calculating the average execution price over a certain time period irrespective of the executed quantity.

The TWAPOrder is based on the AdaptiveOrder (see) below and uses its pricing logic. The TWAPOrder has no additional order properties in addition to the ones defined by the AdaptiveOrder.

When a TWAPOrder is either fully-executed or cancelled a message containing the average price, the benchmark price as well as the execution duration and number of executions is logged to the console. As the reporting functionality needs historical data a historical data adapter will need to be enabled, see Section 7.2.7, "Historical Data Service". The TWAPOrder can still be used without historical data but not report will be logged to the console.

A TWAPOrder can be created and sent as follows:

```java
TWAPOrder order = new TWAPOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setDuration(600)
order.setSliceLength(10)
order.setCancelTime(0.5)
order.setTimeRand(0.25)
order.setQtyRand(0.25)
order.setIncrement(0.05)
order.setInitialOffset(0.8)
order.setMinOffset(0.05)
order.setMaxOffset(1.0)

getOrderService().sendOrder(order);
```

---

2 https://en.wikipedia.org/wiki/Time-weighted_average_price
Alternatively, Section 17.2.1, “Order Preferences” can be used to create a TWAPOrder. The AlgoTrader sample data contains an OrderPreference named TWAP (with the default values shown in the table above) which allows placing a TWAPOrder as follows:

```java
Order order = getOrderService().createOrderByOrderPreference("TWAP");
order.setStrategy(strategy);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);

getOrderService().sendOrder(order);
```

AdaptiveOrder

The AdaptiveOrder is the parent class of the VWAPOrder and TWAPOrder and defines the pricing logic for those. However, it is not possible to send an AdaptiveOrder directly. The AdaptiveOrder has the following order properties.

**Table 24.3. AdaptiveOrder**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Unit</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>startTime</td>
<td>start time of the algo</td>
<td>ZonedDateTime</td>
<td></td>
</tr>
<tr>
<td>endTime</td>
<td>end time of the algo</td>
<td>ZonedDateTime</td>
<td></td>
</tr>
<tr>
<td>duration</td>
<td>duration of the algo</td>
<td>seconds</td>
<td>600</td>
</tr>
<tr>
<td>minSliceQty</td>
<td>minimum child order quantity</td>
<td>BigDecimal</td>
<td></td>
</tr>
<tr>
<td>maxVolPct</td>
<td>maximum % of volBid / volAsk to take</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>sliceLength</td>
<td>average child order length</td>
<td>seconds</td>
<td>10</td>
</tr>
<tr>
<td>cancelTime</td>
<td>% of sliceLength when a child order gets cancelled</td>
<td>%</td>
<td>50%</td>
</tr>
<tr>
<td>timeRand</td>
<td>sliceLength and cancelTime randomization</td>
<td>%</td>
<td>25%</td>
</tr>
<tr>
<td>qtyRand</td>
<td>child order quantity randomization</td>
<td>%</td>
<td>25%</td>
</tr>
<tr>
<td>increment</td>
<td>price increment/decrement</td>
<td>%</td>
<td>5%</td>
</tr>
<tr>
<td>initialOffset</td>
<td>initial offset in % of the first child order</td>
<td>%</td>
<td>80%</td>
</tr>
<tr>
<td>minOffset</td>
<td>minimum offset</td>
<td>%</td>
<td>5%</td>
</tr>
<tr>
<td>maxOffset</td>
<td>maximum offset</td>
<td>%</td>
<td>100%</td>
</tr>
</tbody>
</table>

The AdaptiveOrder uses a pricing logic similar to the Slicing Execution Algo.

The first child order will be placed at the initialOffset between the Bid and the Ask (e.g. at 80%). Depending on whether a child order gets filled, the price of the next child order is adjusted. If the previous child order got fully or partially filled, the price of the next child order will be reduced by increment % of the spread. If the previous child order did not get filled, the price of the next child order is increased by increment % of the spread. The limit price will always be adjusted within the following price range:
The AdaptiveOrder will execute over a predefined time period which can be set by two of the following arguments: startTime, endTime or duration.

New child orders will be sent in randomized time intervals:

Between sliceLength*(1-timeRand) and sliceLength*(1+timeRand)

In case a child order is not fully executed it will get cancelled after the following period of time:

Between sliceLength*cancelTime*(1-timeRand) and sliceLength*cancelTime*(1+timeRand)

The quantity of each child order is randomized in the following interval

between sliceQty*(1-qtyRand) and sliceQty*(1+qtyRand)

Calculated child order quantities respect the optional minSliceQty. In addition, the AdaptiveOrder also respects the optional property maxVolPct which will cause to algo not to place child orders larger than the current VolAsk (for Buy orders) or VolBid (for Sell orders).

As the Algo needs to be executed within a predefined time period, the child order quantities are adjusted throughout the order execution. Quantity adjustments take into consideration previously executed quantity in order to fully executed the algo within its time constraints. No further quantity adjustments take place once 90% of the order execution time has passed.

TargetPositionOrder

The TargetPositionOrder seeks to bring the actual position to an intended target quantity. The TargetPositionOrder starts off by looking up the actual position quantity, calculating the delta between the actual and target quantity and issuing a market order to fill the difference. In many cases the TargetPositionOrder differs little from sending a simple market order. Orders can take some time to fully execute. In the meantime the target position may change. The target quantity of a TargetPositionOrder can be altered at any point of time which will cause the order to re-evaluate its actual state and cancel or modify currently pending order and issue a new order if necessary to match the expected target position. The order also reacts intelligently to stray fills that can occur.

By default TargetPositionOrder is considered to be fully executed once its target position has been reached. The order is then removed from the order book. Often however strategies might want to maintain a particular position over a longer period of time. TargetPositionOrder can be issued with keepAlive attribute set to true to make the order active until explicitly canceled. The order will transition into Status#TARGET_REACHED state once fully executed and will stay there until the target is adjusted or the order is canceled. A TargetPositionOrder can be created and sent as follows:

```java
TargetPositionOrder order = new TargetPositionOrder();
order.setStrategy(strategy);
```
Alternatively, OrderPreferences can be used to create a TargetPositionOrder.

**TrailingLimitOrder**

A TrailingLimitOrder submits an order directly to the exchange with a limit price set a fixed distance away from the current market price. The limit price is adjusted relative to the market price when the market moves in favor of the order. The TrailingLimitOrder is typically used when entering a position on an instrument with a Bullish view.

For a BUY order the limit price will be set a specific amount (defined by the trailingAmount parameter) below the current market price. In case the market price rises, the limit price is increased once the specified minimum amount (defined by the increment parameter) is exceeded. If the market price falls, the limit price stays untouched. If the market price falls below the limit price the order will get filled by the exchange (depending on adequate liquidity).

For a SELL order the limit price will be set a specific amount (defined by the trailingAmount parameter) above the current market price. In case the market price falls, the limit price is decreased once the specified minimum amount (defined by the increment parameter) is exceeded. If the market price rises, the limit price stays untouched. If the market price rises above the limit price the order will get filled by the exchange (depending on adequate liquidity).

A TrailingLimitOrder can be created and sent as follows:

```java
TrailingLimitOrder order = new TrailingLimitOrder();
order.setStrategy(strategy);
order.setAccount(account);
order.setSecurity(security);
order.setQuantity(orderQuantity);
order.setSide(Side.BUY);
order.setTrailingAmount(BigDecimal.valueOf(0.5));
order.setIncrement(BigDecimal.valueOf(0.1));
```

Alternatively, OrderPreferences can be used to create a TrailingLimitOrder.

### 24.2. Execution Algos Retry and Back-off policies

AlgoTrader provides an automated retry handling for Orders sent by Execution Algos (this feature is currently only available for AdaptiveOrders (TWAPOrder and VWAPOrder).
All child orders of TWAP/VWAP Order will be retried in case of:

- Child order send - exception
- Child order send - status REJECTED
- Child order cancel - exception
- Child order cancel - status REJECTED

When the number of retries has exceeded the defined retry limit (default: 3 retries for each order), the entire parent Order (all children Orders) will be cancelled.

The automated retry handling is depicted in the following two diagrams.

![Diagram](image)

**Figure 24.1. Adaptive Send Child Order Retry Policy**
Figure 24.2. Adaptive Cancel Child Order Retry Policy
Synthetic Securities and Derivative Spreads

Figure 25.1. Combinations and Components

AlgoTrader supports Synthetic Securities & Derivative Spreads based on the two Entities Combination and Component.

Combinations are handled like every other Security. A Combination consists of one or many Components. Each component has a quantity.

When trading combinations there are two options:

- **tradable / non-synthetic combinations**
- **synthetic / non-tradable combinations**

For synthetic / non-tradable combinations the AlgoTrader Server generates Ticks based on the size of the components of the combination and the current market values of the associated securities. This calculation is handled by the module `module-combination.epl` which provides the Component Window.

**Note**

It is possible to trade tradable / non-synthetic combinations through the IB interface. For combination orders AlgoTrader will place BAG orders through the IB interface. For this to work it is necessary to have `conids` defined for all components of the combination.

On executions AlgoTrader will create fills for each component and for the combination itself. As a consequence there will be positions on all components as well as the combination itself.
A Combination is available to all strategies and can be subscribed/unsubscribed in the usual manner.

## 25.1. Combination Example

![Table: Market Data](image)

The example above shows a Combination based on 11 ES Mini September 2018 Futures and 3 ES Mini December 2018 Futures. The example shows that the market price of the combination is based on the total prices of both components, e.g. for the ask price:

\[
11 \times 2773 + 3 \times 2777 = 38834
\]

## 25.2. Combination Service

The CombinationService is responsible for handling all Combination / Component related DB-Operations.

### 25.2.1. Create Combination

The following code example shows how to create a combination, add components to it, create a non-tradable position based on it and subscribe to it:

```java
Combination combination = getCombinationService().createCombination(
    CombinationType.RATIO_SPREAD, securityFamilyId);

for (Security security : securities) {
    getCombinationService().addComponentQuantity(
        combination.getId(), security.getId(), quantity);
}

getSubscriptionService().subscribeMarketDataEvent(strategyName, combination.getId());
```

### 25.2.2. Update Component Quantity

The quantity of a Component can be set like this:

```java
getCombinationService().setComponentQuantity(
    combinationSecurityId, componentSecurityId, quantity);
```
To add an amount to the current quantity of a Component:

```java
getCombinationService().addComponentQuantity(
    combinationSecurityId, componentSecurityId, quantity);
```

**Important**

If Components are modified directly in the database, it is necessary to clear the cache as well as to call the method `ServerManagementService.resetComponentWindow` immediately afterwards. If this is not done within a short period of time this might lead to miss-pricing of the corresponding `Combination`. It is therefore preferable to modify Components via the `AlgoTrader Client` or the `CombinationService`.

### 25.2.3. Remove a Component

```java
getCombinationService().removeComponent(combinationSecurityId, componentSecurityId);
```
Chapter 26. CONFIDENTIAL

Spring Services

26.1. Starter Classes

AlgoTrader provides the following starter classes to start up the system for the various operational modes:

- **Reference Data Starters**
  
  When downloading reference data (see Chapter 20, Reference Data) you need to have the following profiles active: singleDataSource or pooledDataSource and the profile of the adapter you want to get reference data from (see table below).

  *You can only have one adapter reference data profile enabled at a time.*

- **Historical Data Starters**
  
  When downloading historical data (see Section 19.3, “Historical Data Download”) you need to have the following profiles active: singleDataSource or pooledDataSource, influxDB and the profile of the adapter you want to get historical data from (see table below).

  *You can only have one adapter historical data profile enabled at a time.*

- **Simulation Starter**
  
  To run a back-test (see Chapter 5, Strategy Backtesting), you need to have the following profiles active: any dataSource profile, simulation. If you are using InfluxDB for back testing, you also need to add influxDB.

- **Embedded Strategy Starters**
  
  When running strategy in embedded mode (see Section 3.2.1, “Embedded Mode”), you need to have the following profiles active: singleDataSource or pooledDataSource, live, embeddedBroker and html5 (if you want to see/use the UI) and the market data, trading profiles.

  If account data (see Chapter 21, Account Data) is required you also need the account profile (see table below).

  The system can be run with several market data, trading and account profiles in the same process.

  If historical data (see Chapter 19, Historical Data) is required you also need one historicalData profile (see table below) and influxDB. If you do not have a historical data provider but still want to store and retrieve historical data using InfluxDB, you need to set `noopHistoricalData` in addition to influxDB.

- **Server Starters**
  
  When running the AlgoTrader server in distributed mode (see Section 3.2.2, “Distributed Mode”), you need to have the following profiles active: singleDataSource or pooledDataSource, live, embeddedBroker and html5 (if you want to see/use the UI) and the market data, trading profiles.
If account data (see Chapter 21, Account Data) is required you also need the account profile (see table below).

The system can be run with several market data, trading and account profiles in the same process.

If historical data (see Chapter 19, Historical Data) is required you also need one historicalData profile (see table below) and influxDB. If you do not have a historical data provider but still want to store and retrieve historical data using InfluxDB, you need to set noopHistoricalData in addition to influxDB.

• Strategy Starters

For strategies running in distributed mode (see Section 3.2.2, “Distributed Mode”), it is sufficient to activate the live Spring profile. In order for a strategy to use historical data, reference data or account related information the profiles historicalData, referenceData and account have to be enabled in on the strategy.

• Reset Starter

The ResetStarter can be used to reset the state of the database to a pre-defined state either before a simulation or if a reset of live trading is required. For parameters and details, see Section 7.2.21, “Reset Service”

• Restore Portfolio Value Starter

The RestorePortfolioValueStarter can restore Portfolio Values for a specified strategy and time period (see Section 7.2.12.2, “Portfolio Value Restoration Feature”)

26.2. Spring Profiles

AlgoTrader is heavily relying on Spring Profiles to activate/deactivate various parts of the system based on user requirements.

General Profiles

• simulation: Contains Spring Beans that are used for Back Tests or when using the Exchange Simulator, e.g. SimulationExecutor, SimulationOrderService and ResetService

• live (client side): Contains Spring Beans needed by Strategies in Live Trading: Esper Engine, LifecycleManager, CacheManager & LookupService

• live (server-side): Contains Spring Beans needed by the Server in Live Trading mode: e.g. Esper Engine

• noopHistoricalData: a no-operation HistoricalDataService. This profile is needed for cases where you want to store/retrieve historical data in InfluxDB but no historical data adapter is active.

• noopHistoricalGenericEvents: a no-operation HistoricalGenericEventsService. This profile is needed for cases where you want to store/retrieve generic data in InfluxDB but no historical data adapter is active.
• **embeddedBroker**: embedded ActiveMQ broker, which is required for sending messages to the UI and to strategies running in distributed mode

• **html5**: the AlgoTrader UI

• **influxDB**: influxDB interface

• **pythonIntegration**: Activates PythonStrategyService that serves to enable communication between AlgoTrader and strategies written in Python, see Section 4.5, “Strategy Development in Python”. It can only be used with SimulationStarter and EmbeddedStrategyStarter

• **noOrderReconciliation**: When activated disables in-built order reconciliation mechanism, see Section 23.5, “Automatic order reconciliation after re-connect”

• **inboundFix**: Enables inbound FIX API, see Section 22.4, “Inbound FIX API”

Additional services profiles (used only by strategy running in distributed mode)

• **historicalData**: means the strategy will require historical data service

• **referenceData**: means the strategy will require reference data service

• **account**: means the strategy will require account service

• **genericEventsService**: means the strategy will require explicit subscription to generic events service

Data Sources: only one data source can be configured

• **pooledDataSource**: c3p0 Pooled Data Source (typically used in live trading both in embedded and distributed mode)

• **singleDataSource**: Spring Driver Manager Data Source (typically used by the Reference Data Starter and Historical Data Starter)

• **hybridDataSource**: Data Source that propagates only reference data to the DB hybridDataSource (typically used in backtesting)

Adapters

**Table 26.1. Adapter Spring Profiles**

<table>
<thead>
<tr>
<th>Adapter</th>
<th>Trading</th>
<th>Market Data</th>
<th>Historical Data</th>
<th>Reference Data</th>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2C2</td>
<td>b2C2Fix</td>
<td>b2C2MarketData</td>
<td>b2C2ReferenceData</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binance</td>
<td>bNC</td>
<td>bNCMarketData</td>
<td>bNCReferenceData</td>
<td></td>
<td>nCAccount</td>
</tr>
<tr>
<td>Bitfinex</td>
<td>bFX</td>
<td>bFXMarketData</td>
<td>bFXReferenceData</td>
<td></td>
<td>FXAccount</td>
</tr>
<tr>
<td>Bitflyer</td>
<td>bFL</td>
<td>bFLMarketData</td>
<td>bFLReferenceData</td>
<td></td>
<td>FLAccount</td>
</tr>
<tr>
<td>BitHumb Pro</td>
<td>bHB</td>
<td>bHBMarketData</td>
<td>bHBRferenceData</td>
<td></td>
<td>HSAccount</td>
</tr>
<tr>
<td>BitMEX</td>
<td>bMX</td>
<td>bMXMarketData</td>
<td>bMXReferenceData</td>
<td></td>
<td>MXAccount</td>
</tr>
<tr>
<td>Bitstamp</td>
<td>bTSFix</td>
<td>bTSMarketData</td>
<td>bTSSReferenceData</td>
<td></td>
<td>TSAccount</td>
</tr>
<tr>
<td>Adapter</td>
<td>Trading</td>
<td>Market Data</td>
<td>Historical Data</td>
<td>Reference Data</td>
<td>Account</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>------------------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Bittrex</td>
<td>bTX</td>
<td>bTXMarketData</td>
<td>bTXReferenceData</td>
<td>bTXAccount</td>
<td></td>
</tr>
<tr>
<td>Blockfills</td>
<td>bFFix</td>
<td>bFMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bloomberg</td>
<td>tDBFix</td>
<td></td>
<td>bBHistoricalData</td>
<td>bBReferenceData</td>
<td></td>
</tr>
<tr>
<td>CoinAPI</td>
<td>cNPMarketData</td>
<td>cNPHistoricalData</td>
<td>cNPReferenceData</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coinbase</td>
<td>cNBFix</td>
<td>cNBMarketData</td>
<td>cNBReferenceData</td>
<td></td>
<td>cNBAccount</td>
</tr>
<tr>
<td>CoinMarketCap</td>
<td></td>
<td></td>
<td></td>
<td>cMCHistoricalData</td>
<td>cMCReferenceData</td>
</tr>
<tr>
<td>CryptoFinance</td>
<td>cFFix</td>
<td>cFMarketData</td>
<td>cFReferenceData</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Currenex</td>
<td>cNXFix</td>
<td>cNXMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deribit</td>
<td>dRBFix</td>
<td>dRBMmarketData</td>
<td>dRBReferenceData</td>
<td>dRBAccount</td>
<td></td>
</tr>
<tr>
<td>DukasCopy</td>
<td>dCFix</td>
<td>dCMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enigma</td>
<td>eGMFix,eGM</td>
<td>eGMMarketData</td>
<td>eGMReferenceData</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eToroX</td>
<td>eTX</td>
<td>eTXMarketData</td>
<td>eTXReferenceData</td>
<td>eTXAccount</td>
<td></td>
</tr>
<tr>
<td>Exante</td>
<td>xNTFix</td>
<td>xNTMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EzeSoft / RealTick</td>
<td>rTFix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortex</td>
<td>fTXFix</td>
<td>fTXMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FXCM</td>
<td>fXCMFix</td>
<td>fXCMMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gemini</td>
<td>gMNFix</td>
<td>gMNFixMarketData</td>
<td></td>
<td>gMNAccount</td>
<td></td>
</tr>
<tr>
<td>Huobi Spot</td>
<td>hBI</td>
<td>hBIMarketData</td>
<td>hBReferenceData</td>
<td>hBIAccount</td>
<td></td>
</tr>
<tr>
<td>InteractiveBrokers</td>
<td>iBNative</td>
<td>IBMarketData</td>
<td>iBHistoricalData</td>
<td>iBReferenceData</td>
<td>iBAccount</td>
</tr>
<tr>
<td>Intrinio</td>
<td></td>
<td>iINTRDividendGenericEvents</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPMorgan</td>
<td>jPMFix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kraken Spot</td>
<td>kKS</td>
<td>kKSMarketData</td>
<td>kKSReferenceData</td>
<td>kKSAccount</td>
<td></td>
</tr>
<tr>
<td>LMAX Digital</td>
<td>lMAXDFix</td>
<td>lMAXDMarketData</td>
<td>lMAXDReferenceData</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMAX Global</td>
<td>lMAXGFix</td>
<td>lMAXGMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nexus Prime</td>
<td>nXSFix</td>
<td>nXSMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OKCoin</td>
<td>oKcoin</td>
<td>oKCoinMarketData</td>
<td>oKCoinReferenceData</td>
<td>oKCoinAccount</td>
<td></td>
</tr>
<tr>
<td>OKEx</td>
<td>oKEX</td>
<td>oKEXMarketData</td>
<td>oKEXReferenceData</td>
<td>oKEXAccount</td>
<td></td>
</tr>
<tr>
<td>One Zero</td>
<td>oZFix</td>
<td>oZMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrimeXM</td>
<td>pXMFix</td>
<td>pXMMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quandl</td>
<td></td>
<td></td>
<td>qDLHistoricalData</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This adapter only provides dividend data

<table>
<thead>
<tr>
<th>Adapter</th>
<th>Trading</th>
<th>Market Data</th>
<th>Historical Data</th>
<th>Reference Data</th>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuantHouse</td>
<td>qHMarketData</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refinitiv</td>
<td>refinitiv</td>
<td>refinitivMarketData</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SocGen</td>
<td>sGFix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trading Technologies</td>
<td>tTFix</td>
<td>tTMarketData</td>
<td></td>
<td>tTReferenceData</td>
<td></td>
</tr>
<tr>
<td>UBS</td>
<td>uBSFix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note

Please append MarketData, HistoricalData, ReferenceData or Account to the Spring Profiles listed above. Example: The Bloomberg market data profile is bBMarketData instead of just bB.

All other services not mentioned above are active in all profiles.

To enable a Profile on start-up, the following VM argument has to be used:

-Dspring.profiles.active=iBMarketData,iBNative
Chapter 27.

Configuration and Preferences API

27.1. Configuration Files

The AlgoTrader Server contains the following two main configuration files.

conf.properties is the main public configuration file:

- Dataset Configuration
- Simulation Settings
- Reporting Settings
- Order / Execution Settings
- RMI Settings
- ActiveMQ Settings
- Jetty Settings

conf-core.properties contains settings that are only used by the core project:

- Data Source Configuration
- Server Engine module definition
- Esper Statements
- Hedging Settings
- ActiveMQ Settings
- Jetty Settings
- SSL Settings
- HttpClient Settings
- Mail Settings
- AlgoTrader UI Settings

In addition adapters may have their own settings file. e.g. conf-ib.properties for IB and conf-bb.properties for BB.
Configuration parameters can be changed inside the .properties files. As an alternative configuration parameters can be provided as VM Options in which case they will overwrite existing parameters inside *.properties files.

```
-Dstatement.closePosition=false
```

Most configuration parameters are prefixed with a namespace (e.g. dataSource, simulation, statement, misc, etc.)

### 27.1.1. Encrypting sensitive configuration values

For security reasons, it is recommended to store sensitive configuration like adapter API key and API secret, in encrypted form.

In Docker based installations the recommended method for that is using **Docker Secrets**

For non-Docker installations it is possible to encrypt property value using **Jasypt** command line utility:

```
...\jasypt-1.9.2\bin>encrypt password="mypassword" input="revdPMxxxxxxxxwCPo"
```

The encrypted value should be copied to relevant property file, e.g. to `conf-bmx.properties`. Alternatively the properties can be changed via **Section 2.4, "VM Options"**:

```
#{"type":"String","label":"API Key"}
bmx.apiKey=ENC(Gv5bH18YmbavDnC3DECMKTOh7wRq5VuKpeNo5tYmaALkpJw0ApEMA==)
```

The steps above should be done for each sensitive value (e.g. apiKey, apiSecret) of each adapter.

In addition the following VM argument must be set to enable encryption

```
-Dconfig.encryption=true
```

Each time the system starts, the user will be prompted to enter the password.

### 27.2. Esper Variables

The configuration files are also used to define values for Esper variables. Because the Esper Variable system is strong typed, variables with their type have to be configured within the corresponding Esper configuration files. e.g.

```
<variable name="simulation_eventsPerDay" type="long"/>
```

### Notes

1. [http://www.jasypt.org/cli.html](http://www.jasypt.org/cli.html)
Note

Name spaces have to be specified using an underscore instead of a period. e.g. `simulation_eventsPerDay` corresponds to `simulation.eventsPerDay` in the property file.
Processes and Networking

28.1. SSL security

By default AlgoTrader is assumed to be running within a secure network segment wherein user authentication and authorization as well SSL security are enforced by the runtime environment / operating system. The AlgoTrader process, individual strategy process and browsers running the HTML5 UI exchange data unencrypted primarily to avoid overhead of encryption for maximal performance.

SSL security can be activated through the following property in conf.properties. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```
# TLS/SSL transport security
ssl.enabled = true
```

By default AlgoTrader ships with a self-signed certificate which can be import into the browser. Please note that modern browsers will show a warning when using self-signed certificates due to your domain name being different from AlgoTrader's own domain.

It is therefore strongly recommended to procure a certificate from a major CA (certification authority) trusted by common browsers. Alternatively you can create your own self-signed certificate for testing purposes, the following command will created a certificate for the domain xxx.algotrader.com

```
```

To use SSL security please update the following properties in conf.properties. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```
# Keystore with SSL key
ssl.keystore = classpath://identity.jks

# Keystore type (JKS will be assumed by default)
ssl.keystoreType =

# Keystore password
ssl.keystorePassword = password

# Private key password
ssl.keyPassword = password
```

When running with TLS transport security turned on AlgoTrader also enforces BASIC user authentication with a user name and a password when logging into the HTML5 UI. User credentials can be provided in conf.properties. Alternatively the properties can be changed via Section 2.4, “VM Options”:
# Web UI user name
jetty.user = myusername

# Web UI password
jetty.password = secret

28.1.1. Importing Certificate into Chrome Browser

1. On the page with the untrusted certificate, click **Ctrl-Shift-I** to open Developer Tools and go to Security
2. Click View Certificate / Details tab > Copy to File. Choose DER encoded binary (.CER)
   - On MacOS drag&drop certificate icon to Finder window
3. Open up Chrome Settings > Show advanced settings > HTTPS/SSL > Manage Certificates.
4. Import the exported .CER file, save into "Trusted Root Certificate Authorities"
5. Check all boxes and click OK. Restart Chrome.
Metrics

In Simulation Mode the performance objective of the system is high-throughput, whereas in Live Trading Mode the objective is low latency. To pinpoint potential performance bottlenecks, AlgoTrader has a built-in metrics functionality.

29.1. Esper Engine Metrics

Metrics for Esper Engines contain statement execution times and event counts.

To enable this feature set `metrics.engine.enabled` to `true` in `conf.properties`.

Enabling this flag will cause additional `metrics` module to be deployed on `SERVER` strategy.

```
protected void onInit(final LifecycleEventVO event) {
    getEngine().deployModule("metrics");
}
```

Note

If your strategy is using Esper modules you want to collect metrics from, you should deploy `metrics` module to your strategy engine:

Metrics will be collected and logged at configurable interval. It can be adjusted by `metrics.engine.logging.intervalSeconds` parameter in `conf.properties`.

By default only resource-heavy statements will be logged - filtered by mean statement execution time. Statements with mean execution time below the threshold will not be logged. The threshold can be configured by `metrics.engine.logging.thresholdMs` parameter in `conf.properties`.

Note

Subscriber time consumption is not included in statement metrics, whereas static method invocation is included.
Logging

AlgoTrader logging is provided by Apache Log4j \(^1\) framework. The Logging system is configured by means of log4j2.xml file.

The log level can be changed through the following VM argument:

```
-DlogLevel=ERROR
```

### 30.1. log4j2.xml

Table 30.1. Default Log4j Appenders

<table>
<thead>
<tr>
<th>Appender</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StdOut</td>
<td>Logs to Standard Out</td>
</tr>
<tr>
<td>StdErr</td>
<td>Logs to Standard Error</td>
</tr>
<tr>
<td>LogEvent</td>
<td>Custom UI appender. Sends log messages to UI</td>
</tr>
</tbody>
</table>

### 30.2. Production log4j2.xml

For production usage it is recommended to adapt the log4j2.xml to client specific needs. Additional samples for production use are available inside log4j2.xml.

Table 30.2. Production Log4j Appenders

<table>
<thead>
<tr>
<th>Appender</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Logs to an appending file</td>
</tr>
<tr>
<td>Mail</td>
<td>Sends Email Messages on Errors</td>
</tr>
</tbody>
</table>

**Note**

- Problems with the Email Appender go to `System.err` (on server see `nohup.log`)
- To prevent saturation of the logs several loggers have been defined with a logging level higher than the root log level

Detailed description of Log4j2 appenders and advanced configuration can be found at the Apache Logging\(^2\) site.

---

\(^1\) http://logging.apache.org/log4j/2.x/

\(^2\) http://logging.apache.org/log4j/2.x/manual/configuration.html
AlgoTrader provides a convenient way to create custom CSV reports for strategy specific reporting. All relevant classes are available inside the package ch.algotrader.report.

To use the reporting functionality create a class similar to this:

```java
public class MyCustomReport {

    private ListReporter reporter;

    public OrderReport() {
        String[] header = new String[] { "Date", "Symbol", "Quantity", "Signal" };
        this.reporter = new ListReporter(Report.generateFile("OrderReport"), header);
    }

    public void write(Date date, String symbol, int quantity, String signal) {
        this.reporter.write(date, symbol, quantity, signal);
    }
}
```

This will create a .csv report named OrderReport.csv inside the directory /files/report/ which contains the columns Date, Symbol, Quantity, Signal.

Some Reports are available out-of-the-box, for further details please see Section 5.5, “Performance Statistics”
Monitoring

32.1. Technical monitoring

Available under our the /monitoring\(^1\) endpoint, users can find a comprehensive monitoring tool that exposes various important metrics both real-time and historical. It allows you review:

- Memory consumption
- CPU usage
- JDBC connections in use
- SQL queries per minute
- HTTP calls per minute
- etc

\(^1\) localhost:9090/monitoring
You can also find detailed information on response time of your operations for example order requests:

**Statistics http - 1 day since midnight**

<table>
<thead>
<tr>
<th>Request</th>
<th>% of cumulative time</th>
<th>Hits</th>
<th>Mean time (ms)</th>
<th>Max time (ms)</th>
<th>Standard deviation</th>
<th>% of cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>http global</td>
<td>100</td>
<td>4,081</td>
<td>14</td>
<td>2,653</td>
<td>63</td>
<td>100</td>
</tr>
<tr>
<td>http warning</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>http severe</td>
<td>10</td>
<td>6</td>
<td>1,027</td>
<td>2,653</td>
<td>1,197</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request</th>
<th>% of cumulative time</th>
<th>Hits</th>
<th>Mean time (ms)</th>
<th>Max time (ms)</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/store/order POST</td>
<td>85</td>
<td>3,910</td>
<td>12</td>
<td>387</td>
<td>31</td>
</tr>
<tr>
<td>/store/order/confirm GET</td>
<td>7</td>
<td>3</td>
<td>3,359</td>
<td>2,632</td>
<td>1,199</td>
</tr>
<tr>
<td>/store/order/exchange GET</td>
<td>3</td>
<td>3</td>
<td>699</td>
<td>2,653</td>
<td>1,172</td>
</tr>
<tr>
<td>/store/order/id GET</td>
<td>1</td>
<td>22</td>
<td>34</td>
<td>42</td>
<td>50</td>
</tr>
<tr>
<td>/store/security/id GET</td>
<td>0</td>
<td>38</td>
<td>8</td>
<td>18</td>
<td>10</td>
</tr>
<tr>
<td>/store/exchange GET</td>
<td>0</td>
<td>3</td>
<td>76</td>
<td>122</td>
<td>64</td>
</tr>
<tr>
<td>/store/exchange/orderstatus GET</td>
<td>0</td>
<td>3</td>
<td>32</td>
<td>69</td>
<td>32</td>
</tr>
<tr>
<td>/store/account GET</td>
<td>0</td>
<td>3</td>
<td>23</td>
<td>40</td>
<td>20</td>
</tr>
</tbody>
</table>

On top of these there are also a set of admin functionalities exposed. You would typically need to install different tools to perform tasks such as reviewing OS processes or taking a snapshot of the heap memory.
32.2. Health check

There is a way to see health state of AlogTrader components, e.g. market adapters, database, Hazelcast caches, JVM

This information is exposed via REST API and as a web page. Example of the health check page:

![Health check results](image)

**Figure 32.1. Health check results**

Description of columns is the following:

<table>
<thead>
<tr>
<th>Column name on the web page/JSON</th>
<th>Description</th>
<th>Status of the component</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA/node name</td>
<td>name of the component</td>
<td>ALIVE/UNHEALTHY/DEAD</td>
</tr>
<tr>
<td>Message/msg</td>
<td>free form comment regarding the status of the component</td>
<td></td>
</tr>
<tr>
<td>Status/status</td>
<td>one of the following values: ALIVE/UNHEALTHY/DEAD</td>
<td></td>
</tr>
</tbody>
</table>

Example of the result as JSON map:

```json
{
  "JVM": {
    "msg": "HeapMemoryUsage: init = 327155712(319488K) used = 497207792(485554K) committed = 838336512(818688K) max = 4632608768(4524032K), NonHeapMemoryUsage: init = 2555904(2496K) used = 229961040(224571K) committed = 238157824(232576K) max = -1(-1K), ThreadCount: 128, Uptime: 9640403ms, GcCollectionTime: 24082ms, GcCollectionCount: 659", "status": "ALIVE"
  },
  "ComboPooledDataSource": {
    "msg": "Connection is valid",
    "status": "ALIVE"
  },
  "HazelcastInstance": {
    "msg": "HazelcastInstance is running"
  }
}
```
Rest endpoints:

- `/util/health`: returns JSON map with service name mapped to its health state
- `/util/health/html`: the same as above but result is a simple HTML table

UI which automatically refresh health status of all components and provides the same information, URL: http://localhost:9090/health.html

Health state is published via ActiveMQ and STOMP, message topics are (for different status levels - alive, unhealthy, dead):

```
/topic/health-status.dead, example message:
   {"status":"DEAD","msg":"no status updates yet","serviceName":"CNB-session"}
```

```
/topic/health-status.unhealthy, example message:
   {"status":"UNHEALTHY","msg":"connected","serviceName":"CNB-session"}
```

```
/topic/health-status.alive, example message:
   {"status":"ALIVE","msg":"loggedIn","serviceName":"CNB-session"}
```

Note: all health statuses are propagated to the strategy via method described below:
public class StrategyService implements ... HealthStatusListener ...
...
@override
public void onHealthStatusChanged(final HealthStatusVO healthStatus) {
}
...
Appendix A. Example Strategy "BreakOut"

A.1. Trading Idea

Warning
The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Strategy trades the EUR.USD FX Market and is based on a simple Breakout Indicator.

The Strategy opens a long (short) position when the current price exceeds (falls below) the maximum (minimum) of the last n bars. After a new position is opened, a profit target price is set as well as a stop loss. If either profit target or stop is reached, the position is closed. If neither stop nor profit target is reached until the end of n-bars, the position is closed.

Positions are sized based on a defined leverage and the current Net Liquidation Value. All Orders are placed as Market Orders. The initial account size is EUR 1'000'000.

A.2. Example

The following 5-min bar chart gives an example of the BreakOut strategy. At 10:20 an aggregation of the last 5 bars between 09:55 and 10:20 is created, based on which the upper limit at 1113.85 and the lower limit at 1110.53 are calculated. At 10:22:37 the upper limit is crossed for the first time and a long position is entered and both a profit target at 1116.40 and a stop loss at 1111.49 are set automatically. At 10:31:52 the profit target is reached and the position is automatically closed.

Note
This example strategy is a good example of combining a bar based strategy with tick-by-tick based actions. The creation of the upper and lower limits are based on the five 5-min bars but the opening and closing of the position takes place as soon as the limits are reached without waiting for the current bar to finish. This is one of the unique features of AlgoTrader that distinguishes it from other trading platforms that operate exclusively based on bars.
A.3. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, *Strategy Development*.

The following list will give an overview of the specific artifacts implemented by the BreakOut Strategy (Note: Most of the functionality is documented via Javadoc or Esper comments):

/src/main/java/ch/algotrader/strategy/breakOut/BreakOutService.java

The strategy service class providing the main entry method invoked by the Esper ENTRY_LONG and ENTRY_SHORT statements:

/src/main/java/ch/algotrader/strategy/breakOut/BreakOutConfig.java

Contains all strategy configuration items

/src/main/resources/module-breakOut.epl

Esper Module containing all statements for this strategy:

- **INSERT_INTO_BAR**: Creates High/Low Bars
- **INSERT_INTO_BOUND**: Calculates minimum and maximum of last n bars
- **ON_BOUND_SET_TRIGGERS**: sets the upperTrigger and lowerTrigger based on the minimum and maximum of the last n bars
- **ENTRY_LONG / ENTRY_SHORT**: open position if last tick is higher (lower) than previous n bars.
• **CLOSE_LONG_POSITION** / **CLOSE_SHORT_POSITION**: Close position if last tick is higher (lower) than target or lower (lower) than stop

• **CLOSE_OPEN_POSITION**: Close position if neither target nor stop are reached before the end of n-bars

/srC/main/resources/breakOut-default.properties
Contains default parameters used by the strategy (e.g. `lengthOfBar` and `numberOfBars`)

/srC/main/resources/META-INF/esper-breakOut.cfg.xml
Contains event-types definitions (i.e. `CurrentValue`), variables (e.g. `lengthOfBar` and `numberOfBars`).

/srC/main/resources/META-INF/applicationContext-client-breakOut.xml
Contains the Spring Bean definitions for `breakOutConfigParams`, `breakOutConfig`, `breakOutEngine`, `breakOutService`.

/srC/main/resources/db/mysql/mysql-breakout.sql
Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

To start the Strategy please see the explanations in *Chapter 3, Starting AlgoTrader*.

### A.4. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

**Git Clone**

If you have used the *Section 2.1, “Windows Installer”* to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```
git clone https://gitlab.algotrader.com/general/examples.git
```

Import the project `breakOut` into your IDE.

**Tick Data File**

You will find a historical tick data file under `breakOut/files/tickdata/eurusd-1min-20111218-20130121/EURUSD.csv`

You can also import this tick data csv file into InfluxDB through the historical data manager (see *Section 11.3, “Historical Data Manager”* Historical Data Import) and run simulations based on the Influx data.

**Start the Simulation**

launch the Run Configuration: `SimulationStarter-simulate-breakOut`

To start the strategy in live trading mode on a development workstation please execute the following steps:
Initialize the database

- load the db-samples script into the MySQL database: /algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql
- load the strategy specific script into the MySQL database: breakOut/src/main/resources/db/mysql/mysql-data.sql

Start the Strategy

- invoke the Run Configuration: EmbeddedStarter-breakOut

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file
- Copy the following file to the server and make changes as needed:
  https://gitlab.algotrader.com/general/examples/blob/master/breakOut/docker-compose.yml

Run docker compose
- Invoke the following command inside the directory where the docker-compose.yml file is located:
  
  ```
  docker-compose up -d
  ```

**Note**

Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside /bootstrap/launch. this will load the MySQL sample data

```
  docker-compose up -d mysql ibgateway algotrader
```
Appendix B. Example Strategy "Box"

B.1. Trading Idea

Warning

The purpose of this Strategy is to demonstrate the capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! Due to frequent Draw Downs, it might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Strategy trades the EUR.USD FX Market and is based on the Stairstep Breakouts (SSBO) Indicator that is presented on www.forexfactory.com\(^1\) by forexhard\(^2\).

The Trading Idea behind the Strategy is the following: Markets will often stay within a trading range for a considerable amount of time before they break-out in either direction. The following Chart shows some examples of trading ranges.

Figure B.1. Box Trading Ranges

After a break-out markets might return back into the trading range but will eventually make a major move in one direction.

According to the defined settings, the Strategy looks for a trading range with a minimum length in Minutes (e.g. 90 Minutes) and a maximum width in Pips (e.g. 30 Pips). The chart below displays a typical trading range in dark blue color.

\(^1\) https://www.forexfactory.com/showthread.php?t=302007
\(^2\) https://www.forexfactory.com/forexhard
Figure B.2. Box Strategy

As soon as trading range has been built according to these parameters, the Strategy waits for the first break-out to happen. The strategy enters the market in the direction of the breakout as soon as a small margin called buffer (dashed red line, e.g. 5 Pips) has been crossed. In the example above, this happened at 10:48.

The Strategy will set a stop at the opposite side of the box (e.g. 1.3618 = 39 Pips) and a target with the same distance (e.g. 1.3544 = 39 Pips).

If the target is reached, the Strategy resets itself and waits for a new Box to build itself.

If the Position gets stopped out (at the opposite side of the Box), The Strategy waits for the next break-out to happen (on the same Box) and enters the market again after the buffer-line has been crossed. This time the size of the position is doubled in order to cover the losses of the first entry, in case the target is reached this time around. Position size is doubled up to a defined maximum. Because of this doubling the system can be categorized as a Martingale Strategy (see Martingale Betting System\(^3\)).

The following State Chart Diagram depicts the different states the Strategy will pass through:

---

\(^3\) https://en.wikipedia.org/wiki/Martingale_%28betting_system%29
The default setting of the Strategy will go up to level 5 which will result in a position size of 16 times the original size. So the individual sizes on the different levels will be: 1, 2, 4, 8 & 16. Each successful series will therefore present a profit of 1 unit. Very often series will be successful on a level that is below the maximum level (e.g. below level 5). However if the Strategy has a loosing set, which will be terminated at the maximum level (e.g. at level 5), there will be a loss of 16 times the original position size.

The Strategy will often have multiple successful series in a row before having one major draw down. A typical performance chart will therefore look like this:

To prevent having open positions over the weekend the Strategy does not create any new boxes after a defined time on Friday (e.g. 4PM). Also, it will terminate a potential ongoing series at a defined time on Friday (e.g. 10PM)
B.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, Strategy Development.

The following list will give an overview of the specific artifacts implemented by the Box Strategy. Most of the functionality is documented via Javadoc or Esper comments:

/src/main/java/ch/algotrader/strategy/box/BoxService.java
   The strategy service class providing the main methods invoked by different Esper statements.

/src/main/java/ch/algotrader/strategy/box/BoxConfig.java
   Contains all strategy configuration items

/src/main/java/ch/algotrader/strategy/box/Box.java
   A POJO class representing all properties of a Box (e.g. top, bottom, startDateTime and endDateTime)

/src/main/java/ch/algotrader/strategy/box/State.java
   A Java Enum representing the different States the Strategy can pass through (INIT, CREATED, LONG, SHORT, FLAT)

/src/main/resources/module-box-init.epl
   Esper Module containing statements for capturing market data, creating variables and creating Boxes. In Live Trading, these statements will be deployed before the pre feeding.

/src/main/resources/module-box-run.epl
   Esper Module containing statements that invoke the business actions on the BoxService (entry, takeProfit, closePosition, reverse and terminateSeries). In Live Trading these statements will be deployed after pre-feeding is finished.

/src/main/resources/box-default.properties
   Contains parameters used by the strategy (e.g. boxLength and boxRange)

/src/main/resources/META-INF/esper-box.cfg.xml
   Contains event-types definitions (i.e. CurrentValue), imports (i.e. Box and State), variables (e.g. boxLength and boxRange)

/src/main/resources/META-INF/applicationContext-client-box.xml
   Contains the Spring Bean definitions for boxConfigParams, boxConfig, boxEngine, boxService.

/src/main/resources/db/mysql/mysql-box.sql
   Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

/src/main/resources/html5
   HTML5 and JavaScript files needed for the strategy custom web UI
To start the Strategy please see the explanations in *Chapter 3, Starting AlgoTrader*.

### B.3. Strategy Monitoring

The Box strategy is equipped with an HTML5 custom widget that displays current metrics like State, Units, Upper Target, etc. The custom widget also contains a button to terminate the current series.

![Box HTML5 Custom Widget Example](image)

**Figure B.5. Box HTML5 Custom Widget Example**

**Note**

It might be necessary to fully reload the browser on first startup to show the custom widget using Ctrl + Shift + R.
B.4. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

Git clone

If you have used the Section 2.1, “Windows Installer” to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```bash
git clone https://gitlab.algotrader.com/general/examples.git
```

Import the project box into your IDE.

Bar Data File

You will find a historical bar data file under box/files/bardata/eurusd-1min-20111218-20130121/ EURUSD.csv

You can also import this bar data csv file into InfluxDB through the historical data manager (see Section 11.3, “Historical Data Manager” Historical Data Import) and run simulations based on the Influx data.

Start the Simulation

launch the Run Configuration: SimulationStarter-simulate-box

To start the strategy in live trading mode on a development workstation please execute the following steps:

Initialize the database

load the db-samples script into the MySQL database: /algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql

load the strategy specific script into the MySQL database: /box/src/main/resources/db/mysql/mysql-data.sql

Start the Strategy

invoke the Run Configuration: EmbeddedStarter-box

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file

Copy the following file to the server and make changes as needed:

https://gitlab.algotrader.com/general/examples/blob/master/box/docker-compose.yml

Run docker compose

Invoke the following command inside the directory where the docker-compose.yml file is located:
docker-compose up -d

Note
Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside `/bootstrap/launch`. This will load the MySQL sample data.

docker-compose up -d mysql ibgateway algotrader
Appendix C. Example Strategy "Pairs Trading"

C.1. Trading Idea

**Warning**

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Pairs Trading strategy uses the web service [www.pairtradinglab.com](https://www.pairtradinglab.com) to trade pairs of US equities

C.1.1. What Is Pairs Trading?

Pairs trading is a well-known market neutral trading strategy, that gives traders the ability to profit from practically any market conditions. Whether conditions reflect an uptrend, downtrend, or sideways movement, traders can take advantage of the current market using pairs trading. This type of strategy is typically categorized as a statistical arbitrage trading strategy.

The strategy works by monitoring the performance of two historically correlated securities. When the correlation between those two securities demonstrate a temporary weakness, a pairs trade can be conducted by shorting the outperforming stock and going long on the under performing stock. Basically, one is betting that the spread between the two will converge eventually.

C.1.2. Pair Trading Lab

Pair Trading Lab offers tools to assist in setting up and backtesting a pairs trading portfolio. Along with a database of more than 10 million pre-analyzed pairs, Pair Trading Lab offers the following:

- Advanced online back tester
- Online co-integration analyzer
- Private repository of backtests, studies, and pairs
- Portfolio organizer and portfolio backtester

C.1.3. AlgoTrader - Pair Trading Lab Integration

With the integration between AlgoTrader and Pair Trading Lab, it is possible take advantage of the capabilities of both systems in combination:

---

1 https://www.pairtradinglab.com/
Pair Trading Lab will be used to:

- Create backtests of pairs
- Verify a pair trading idea and inspect the behavior and robustness of pairs
- Test pairs for co-integration
- Search the PTL database of more than 10 million pre-analyzed U.S. market pairs using complex filters
- Create and maintain lists of interesting pairs, rate them, and tag them
- Create, maintain, and backtest portfolios of pair strategies

Then the AlgoTrader - Pair Trading Lab integration can be used to download selected pairs and/or portfolio of pairs from Pair Trading Lab into AlgoTrader where they can then be traded automatically

The AlgoTrader based pairs trading strategy implementation is based on the *Ratio Model*

## C.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in *Chapter 4, Strategy Development*.

The following list will give an overview of the specific artifacts implemented by the Pairs Trading Strategy (Note: Most of the functionality is documented via Javadoc or Esper comments):

```plaintext
/src/main/java/ch/algotrader/strategy/pairstrading/service/PairsTradingService.java
   The strategy service class providing the main trading logic

/src/main/java/ch/algotrader/strategy/pairstrading/csv/CsvImporter.java
   Import utility to download pairs from Pair Trading Lab and configure them in AlgoTrader

/src/main/java/ch/algotrader/strategy/pairstrading/util/PairsTradingConfig.java
   Contains all strategy configuration items

/src/main/java/ch/algotrader/strategy/pairstrading/util/PairsTradingCalc.java
   Contains the logic of the ratio model

/src/main/resources/module-pairstrading.epl
   Esper Module containing all statements for this strategy:

   - PAIR_WINDOW: Contains all current pair definitions
   - SIGNAL_WINDOW: Contains current signals (will be updated on each tick)
```

---

C.3. Installation & Startup

Before using the strategy please execute the following steps:

Pair Trading Lab account sign-up
   Sign up for a free account at Pair Trading Lab.

Create a pair portfolio
   create a pair portfolio and add some pairs.

Extract Portfolio ID
   csvImportPortfolio needs to be extracted from the URL when clicking on pair in the PTL Trader / Portfolio Manager.

---

4 https://www.pairtradinglab.com/portfolio-manager
To start the strategy in live trading mode on a development workstation please execute the following steps:

**Git Clone**

If you have used the Section 2.1, “Windows Installer” to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```
git clone https://gitlab.algotrader.com/general/examples.git
```

Import the project `pairstrading` into your IDE.

**Deploy MySQL data**

Load the file `pairstrading/src/main/resources/db/mysql/mysql-data.sql` into MySQL

**Configure Pair Trading Lab Credentials**

Inside the file `conf-pairstrading.properties` the following items need to be configured. Alternatively the properties can be changed via Section 2.4, “VM Options”:

```
#{"type":"String","required":"false","label":"Pair Trading Lab Portfolio ID"}
csvImportPortfolio = xyz

#{"type":"String","required":"false","label":"Pair Trading Lab Username"}
csvImportUser = user

#{"type":"String","required":"false","label":"Pair Trading Lab Password"}
```
Start the Strategy in Live Trading Mode
launch the Run Configuration: EmbeddedStarter-pairstrading

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file
Copy the following file to the server and make changes as needed:

https://gitlab.algotrader.com/general/examples/blob/master/pairstrading/docker-compose.yml

Configure Pair Trading Lab Credentials
inside the docker-compose.yml file update the VM_ARGUMENTS environment variable and set the correct values for csvImportPortfolio, csvImportUser & csvImportPassword:

VM_ARGUMENTS: "-DcsvImportPortfolio= -DcsvImportUser= -DcsvImportPassword="

csvImportPortfolio needs to be extracted from Pair Trading Lab (see above)

Run docker compose
Invoke the following command inside the directory where the docker-compose.yml file is located:

docker-compose up -d

Note
Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside /bootstrap/launch. this will load the MySQL sample data

docker-compose up -d mysql ibgateway algotrader

C.4. Strategy Monitoring
The Pairs Trading strategy is equipped with a custom Widget available through component's palette on the main Dashboard.
Figure C.2. Pairs Trading HTML5 Custom Widget Example

Note

It might be necessary to fully reload the browser on first startup to show the custom widget using Ctrl + Shift + R.

The HTML5 management page provides the following controls:

- **PairInfo & Pairs**: current pair definitions as downloaded from Pair Trading Lab. `movingAvg` and `standardDev` are calculated on a daily basis (by the Esper statement `UPDATE_HISTORICAL_BARS`) using historical closing prices.

- **Signals**: intraday pair values based on live data. `ratio` shows the current price ratio between individual instruments of a pair. `zScore` shows the current ratio relative to the Bollinger band around the ratio time series. When the `zScore` hits the `zScoreEntry` threshold a position is entered, and when the `zScore` hits the `zScoreExit` threshold the position is closed. If the `zScore` happens to be above `zScoreMax` (e.g. after a large overnight gap) no new position will be opened. The `signal` field shows the current state of a pair (i.e. LONG, SHORT, EXIT & HOLD).
• The action **Import Historical Bars** is used to import historical closing prices of all instruments for the relevant look back period. This action is automatically executed once a day. In addition it can be invoked manually at any time.

• The action **Re-Calc Entry Thresholds** is used to update `movingAvg` and `standardDev` based on historical data in the database. This action is automatically executed once a day. In addition it can be invoked manually at any time.

• The action **Import Pairs** imports and/or update pairs from Pair Trading Lab.
Appendix D. Example Strategy "IPO"

D.1. Trading Idea

Warning

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The strategy trades US equity IPOs (initial public offering). When a new stock is launched on the exchange for the first time the strategy tries to realize trading profits of the first trading day. New IPOs are announced on web pages like IPOScoop\(^1\) together with an indicative open price.

D.2. Strategy Monitoring

The IPO strategy is equipped with the following HTML5 custom widget which displays currently active IPOs and allows adding and removing IPOs while the strategy is running.

\(^1\) https://www.iposcoop.com/
Note

It might be necessary to fully reload the browser on first startup to show the custom widget using Ctrl + Shift + R.

Inside the custom widget the user can enter a new IPO to be traded by populating the following fields:

- **Symbol** to be traded
- **SecurityFamily** the symbol belongs to
- **Limit** The maximum limit price of the initial order
- **Multiplier** to be applied to the opening price for secondary orders (see below)
For each symbol entered, the strategy will place orders at a configurable time in the morning (e.g. 4:30am).

**Note**
Trading of IPOs usually starts within 2-3 hours after the official market open

Definitions:

- Cash commitment = starting capital / number of symbols to be traded
- Quantity per symbol = cash commitment / user-defined limit price

The strategy will place a limit-at-the-open order at user-defined limit price for the entry quantity specified above. Using a limit-at-the-open order will cause the order to participate in the opening auction of the IPO.

Immediately following the open of regular trading, the strategy will check to see if the entire cash commitment for the given symbol has been exhausted.

1. If it has been exhausted, no further action will be taken
2. If the cash commitment has not been exhausted and the stock opened above the **LIMIT PRICE**, the unfilled quantity will be cancelled, and no further action will be taken by the strategy.
3. Otherwise, if the cash commitment for the name has NOT been exhausted a secondary limit order is placed as per below:

   - Remaining cash = Cash commitment - cash used on current position
   - Quantity per symbol = Remaining cash / opening price
   - Limit price = opening price * MULTIPLIER (e.g. 1.02)

If the secondary order is not filled by a configurable end time (e.g. 3:30pm), it will be cancelled by the system.

Immediately after executing any buy orders, the strategy will place a market-on-close order for the entire position.

**D.3. Implementation**

The main artifacts needed for the Implementation of a new Strategy are described in *Chapter 4, Strategy Development*.

The following list will give an overview of the specific artifacts implemented by the IPO Strategy (Note: Most of the functionality is documented via Javadoc or Esper comments):

/src/main/java/ch/algotrader/strategy/ipo/IPOService.java

The strategy service class providing the main trading logic
/src/main/java/ch/algotrader/strategy/ipo/IPO.java

Java POJO class representing a single IPO

/src/main/resources/module-ipo.epl

Esper Module containing all statements for this strategy:

- **SEND_ATO_ORDERS**: sends out at-the-open orders at the configured
- **SEND_LIMIT_ORDERS**: triggers the secondary order service once the at-the-open order has been fully executed and the official open price (via GenericTickVO) has been disseminated. An Esper Join is used for this since either one of those events can arrive first
- **CLOSE_OPEN_ORDERS**: cancels all orders at the configured time
- **DAILY_CLEAN_UP**: unsubscribes all market data and resets the list of IPOs an initial capital

/src/main/resources/conf-ipo.properties

Contains default parameters used by the strategy

/src/main/resources/META-INF/esper-ipo.cfg.xml

Contains Esper variables for the strategy

/src/main/resources/META-INF/applicationContext-client-ipo.xml

Contains ipoConfigParams, ipoEngine & ipoService.

/src/main/resources/db/mysql/mysql-ipo.sql

Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

/src/main/resources/html5

HTML5 and JavaScript files needed for the strategy custom web UI

To start the Strategy please see the explanations in *Chapter 3, Starting AlgoTrader*.

**D.4. Installation & Startup**

To start the strategy in live trading mode on a development workstation please execute the following steps:

**Git Clone**

If you have used the *Section 2.1, “Windows Installer”* to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```
git clone https://gitlab.algotrader.com/general/examples.git
```

Import the project *ipo* into your IDE.

**Deploy MySQL data**

Load the file *ipo/src/main/resources/db/mysql/mysql-data.sql* into MySQL
Start the Strategy in Live Trading Mode
launch the Run Configuration: EmbeddedStarter-ipo

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file
Copy the following file to the server and make changes as needed:

https://gitlab.algotrader.com/general/examples/blob/master/ipo/docker-compose.yml

Run docker compose
Invoke the following command inside the directory where the docker-compose.yml file is located:

docker-compose up -d

Note
Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside /bootstrap/launch. this will load the MySQL sample data

docker-compose up -d mysql ibgateway algotrader
Appendix E. Example Strategy "EMA"

E.1. Trading Idea

Warning

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Strategy is a simple example without Esper that trades the EUR.USD FX Market and is based on two exponential moving averages.

The Strategy sends a BUY order when shorter moving average (e.g. 10-days) crosses above the longer moving average (e.g. 20-days) and it sends a SELL order when shorter moving average crosses below the longer moving average.

![EMA Strategy Example](image.png)

Figure E.1. EMA Strategy Example

E.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, Strategy Development.
The strategy uses the TA4J\(^1\) library which provides over 100 technical indicator that are computed on a continuous basis.

The EMA Strategy consist of one single Java class only:

\[\text{/src/main/java/ch/algotrader/strategy/ema/EMAService.java}\]

The strategy service class providing `onStart` and `onBar` method containing the trading logic.

To start the Strategy please see the explanations in *Chapter 3, Starting AlgoTrader*.

**E.3. Installation & Startup**

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

**Git Clone**

If you have used the *Section 2.1, “Windows Installer”* to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```
  git clone https://gitlab.algotrader.com/general/examples.git
```

Import the project `ema` into your IDE.

**Bar Data File**

You will find a historical bar data file under `ema/files/barData/eurusd-1min-20111218-20130121/EURUSD.csv`

You can also import this tick data csv file into InfluxDB through the historical data manager (see *Section 11.3, “Historical Data Manager”* Historical Data Import) and run simulations based on the Influx data.

**Start the Simulation**

Launch the Run Configuration: `SimulationStarter-simulate-ema`

To start the strategy in live trading mode on a development workstation please execute the following steps:

**Initialize the database**

- load the db-samples script into the MySQL database: `/algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql`
- load the strategy specific script into the MySQL database: `/ema/src/main/resources/db/mysql/mysql-data.sql`

\(^1\) https://github.com/mdeverdelhan/ta4j-origins
Start the Strategy

invoke the Run Configuration: EmbeddedStarter-ema

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file

Copy the following file to the server and make changes as needed:

https://gitlab.algotrader.com/general/examples/blob/master/ema/docker-compose.yml

Run docker compose

Invoke the following command inside the directory where the docker-compose.yml file is located:

docker-compose up -d

Note

Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside /bootstrap/launch. this will load the MySQL sample data

docker-compose up -d mysql ibgateway algotrader
Appendix F. Example Strategy "Random"

F.1. Trading Idea

**Warning**

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The Strategy is a simple example that places random orders at regular intervals. The Random strategy is used for the *AlgoTrader Demo*¹.

F.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in *Chapter 4, Strategy Development*.

The Random Strategy consist of the following artifacts:

/src/main/java/ch/algotrader/strategy/random/RandomService.java
- The strategy service class providing the main methods invoked by different Esper statements.

/src/main/java/ch/algotrader/strategy/random/RandomConfig.java
- Contains all strategy configuration items

/src/main/resources/module-random.epl
- Esper Module containing statements to place and cancel orders as well as update subscriptions once a day.

/src/main/resources/conf-random.properties
- Contains parameters used by the strategy (e.g. `positionMax` and `orderMax`)

/src/main/resources/META-INF/esper-random.cfg.xml
- Contains variables (i.e. `placeOrderInterval` and `cancelOrderInterval`)

/src/main/resources/META-INF/applicationContext-client-random.xml
- Contains the Spring Bean definitions for `randomConfigParams`, `randomConfig`, `randomEngine`, `randomService`.

/src/main/resources/db/mysql/mysql-data.sql
- Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

To start the Strategy please see the explanations in Chapter 3, Starting AlgoTrader.

F.3. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

Git Clone

If you have used the Section 2.1, “Windows Installer” to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```
git clone https://gitlab.algotrader.com/general/examples.git
```

Import the project `random` into your IDE.

To start the strategy in live trading mode on a development workstation please execute the following steps:

Initialize the database

- load the db-samples script into the MySQL database: `/algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql`
- load the strategy specific script into the MySQL database: `/random/src/main/resources/db/mysql/mysql-data.sql`

Start the Strategy

invoke the Run Configuration: `EmbeddedStarter-random`

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file

Copy the following file to the server and make changes as needed:

```
https://gitlab.algotrader.com/general/examples/blob/master/random/docker-compose.yml
```

Run docker compose

Invoke the following command inside the directory where the `docker-compose.yml` file is located:

```
docker-compose up -d
```

Note

Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside `/bootstrap/launch`. this will load the MySQL sample data.
docker-compose up -d mysql ibgateway algotrader
Appendix G. Example Strategy "Spreader"

G.1. Trading Idea

Warning

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

The spreader is a pair trading strategy that seeks to either short or buy the spread between EURUSD@IB and GBPUSD@IB when the value of the spread between the price of both instruments breaches a certain threshold.

The Strategy enters a LONG/SHORT position if the spread goes below the "go long" threshold and enters a SHORT/LONG position if the spread goes over the "go short" threshold.

Positions are closed either with profit (spread moves favourably by the take profit offset) or with loss (spread moves unfavourably by the stop loss offset).

G.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in Chapter 4, Strategy Development.

The Spreader Strategy consist of the following artifacts:

/src/main/java/ch/algotrader/strategy/SpreaderService.java

The strategy service class providing the main methods invoked by different Esper statements.

/src/main/java/ch/algotrader/strategy/SpreaderConfig.java

Contains all strategy configuration items

/src/main/java/ch/algotrader/strategy/State.java

Defines the state the strategy can be in. (LONG, SHORT, FLAT, PENDING_LONG, PENDING_SHORT, PENDING_FLAT)

/src/main/resources/conf-spreader.properties

Contains parameters used by the strategy (e.g. firstSecurityId and secondSecurityId and firstSecurityFeedType and secondSecurityFeedType)

/src/main/resources/META-INF/applicationContext-client-spreader.xml

Contains the Spring Bean definitions for spreaderConfig, spreaderConfigParams, spreaderEngine, spreaderService.
Installation & Startup

385/src/main/resources/db/mysql/mysql-data.sql
Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

To start the Strategy please see the explanations in Chapter 3, Starting AlgoTrader.

G.3. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

Git Clone
If you have used the Section 2.1, “Windows Installer” to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```
git clone https://gitlab.algotrader.com/general/examples.git
```

Import the project spreader into your IDE.

To start the strategy in live trading mode on a development workstation please execute the following steps:

Initialize the database
load the db-samples script into the MySQL database: /algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql

load the strategy specific script into the MySQL database: /spreader/src/main/resources/db/mysql/mysql-data.sql

Start the Strategy
invoke the Run Configuration: EmbeddedStarter-spreader

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file
Copy the following file to the server and make changes as needed:

```
https://gitlab.algotrader.com/general/examples/blob/master/spreader/docker-compose.yml
```

Run docker compose
Invoke the following command inside the directory where the docker-compose.yml file is located:

```
docker-compose up -d
```
Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside `/bootstrap/launch`. This will load the MySQL sample data.

```
docker-compose up -d mysql ibgateway algotrader
```
Appendix H. Example Strategy "Delta Hedge"

H.1. Trading Idea

Warning
The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

Delta hedging is an options strategy that aims to reduce or hedge, the risk associated with price movements in the underlying asset. The approach uses Futures to offset the risk to an entire portfolio of holdings. The investor tries to reach a delta neutral state and not have a directional bias on the hedge.

Since delta hedging attempts to neutralize or reduce the extent of the move in an option's price relative to the asset's price, it requires a constant rebalancing of the hedge.

The Strategy tracks all position changes and calculates deltas for each position every 10 seconds. Every 30 seconds it calculates portfolio average deltas and adjusted market values.

Depending on the results - strategy hedges positions using Futures until it reaches a delta neutral state.

H.2. Implementation

The following list will give an overview of the specific artifacts implemented by the Delta Strategy. Most of the functionality is documented via comments:

/src/main/java/ch/algotrader/strategy/delta/DeltaService.java
The strategy service class providing the all the logic.

/src/main/java/ch/algotrader/strategy/delta/DeltaConfig.java
Contains all strategy configuration items

/src/main/java/ch/algotrader/strategy/delta/DeltaSnapshotVO.java
A POJO class representing current state of security

/src/main/java/ch/algotrader/strategy/delta/DeltaEventVO.java
A POJO class representing the collection of DeltaSnapshotVO containing current state of entire portfolio

This POJO is transported to the UI via Generic Events

/src/main/resources/delta-default.properties
Contains parameters used by the strategy
The Delta strategy is equipped with an HTML5 custom widget that displays current metrics like Delta, Adjusted Market Value, etc.

![Figure H.1. Delta Strategy UI](image)

### H.4. Installation & Startup

To setup the strategy for live trading on a development workstation please execute the following steps:

**Git clone**

If you have used the *Section 2.1, “Windows Installer”* to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:
git clone https://gitlab.algotrader.com/general/examples.git

Import the project delta into your IDE.

Initialize the database
  load the db-samples script into the MySQL database: /algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql
  load the strategy specific script into the MySQL database: /delta/src/main/resources/db.mysql/mysql-data.sql

Download Reference Data
  invoke the Run Configuration for Futures reference data: ReferenceDataStarter-futures-delta
  invoke the Run Configuration for Options reference data: ReferenceDataStarter-options-delta

Start the Strategy
  invoke the Run Configuration: EmbeddedStrategyStarter-delta

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file
  Copy the following file to the server and make changes as needed:

  https://gitlab.algotrader.com/general/examples/blob/master/delta/docker-compose.yml

Run docker compose
  Invoke the following command inside the directory where the docker-compose.yml file is located:

  docker-compose up -d

Note

Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside /bootstrap/launch. This will load the MySQL sample data

  docker-compose up -d mysql ibgateway algotrader
Appendix I. Example Strategy "Short Strangle"

I.1. Trading Idea

Warning

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it Live.

A short strangle is an options strategy comprised of selling both a call option and a put option with the same strike price and expiration date. It is used when the trader believes the underlying asset will not move significantly higher or lower over the lives of the options contracts. The maximum profit is the amount of premium collected by writing the options. The potential loss can be unlimited, so it is typically a strategy for more advanced traders.

Short strangles allow traders to profit from the lack of movement in the underlying asset, rather than having to place directional bets hoping for a big move either higher or lower. Premiums are collected when the trade is opened with the goal to let both the put and call expire worthless. However, chances that the underlying asset closes exactly at the strike price at the expiration is low, and that leaves the short strangle owner at risk for assignment. However, as long as the difference between asset price and strike price is less than the premiums collected, the trader will still make a profit.

Advanced traders might run this strategy to take advantage of a possible decrease in implied volatility. If implied volatility is unusually high without an obvious reason for it being that way, the call and put may be overvalued. In this case, the goal would be to wait for volatility to drop and then close the position for a profit without waiting for expiration.
Figure I.1. Short Strangle

This example strategy demonstrates a typical options trading strategy where the subscribed instruments change throughout the simulation process (i.e. when options expire new options need to be subscribed).

On startup the strategy will subscribe to the underlying S&P Index. Then on the first tick of the S&P Index the strategy will determine Put and Call options with a minimum remaining time to expiration of \( \text{daysToExpirationOnOpen} \).

A standard Strangle strategy will use Put and Call options with the same strike (the current market price). By setting the strategy parameter \( \text{moneyness} \) to a non-zero amount the Put Option will have a strike of \( \text{moneyness} \) below the current market price and the Call option will have a strike of \( \text{moneyness} \) above the current market price. By setting \( \text{moneyness} \) to a non-zero amount the strategy becomes a Short Strangle strategy.

After subscribing to the newly established Put and Call Option the strategy will wait for the first tick of each options and then place corresponding \text{SELL} \ orders.

The strategy parameter \( \text{daysToExpirationOnClose} \) then defines the number of days prior to expiration of the options the strategy will close open positions. Upon closing positions the strategy will unsubscribe the current option market data subscriptions and establish a new set of options to enter positions into.

I.2. Implementation

The following list will give an overview of the specific artifacts implemented by the Short Strangle Strategy. Most of the functionality is documented via comments:
The strategy service class providing the all the logic.

Contains parameters used by the strategy

Contains the Spring Bean definitions for shortstrangleConfigParams, shortstrangleEngine, shortstrangleService

Esper config file (no strategy specific config needed)

Contains the end-of-day historical data for all SPX Options for July 2011 to January 2013

I.3. Installation & Startup

To setup the strategy for live trading on a development workstation please execute the following steps:

**Git clone**

If you have used the Section 2.1, “Windows Installer” to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```bash
git clone https://gitlab.algotrader.com/general/examples.git
```

Import the project short-strangle into your IDE.

**Initialize the database**

Load the InfluxDB historical data file into InfluxDB: /short-strangle/src/main/resources/db/influx/influx-shortstrangle.gz by issuing the following command

```bash
influx -import -compressed -path influx-shortstrangle.gz
```

**Start the Simulation**

Invoke the Run Configuration: SimulationStarter-simulate-shortstrangle
Appendix J. Example Strategy "Dividend Capture"

J.1. Trading Idea

**Warning**

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it live.

The purpose of this strategy is to only illustrate the concept of capturing the dividend.

The idea is to receive a dividend from a company by holding its stock prior to ex-date.

The Strategy enters a LONG position as soon as dividend date is announced and closes the position on ex-date thus obtaining the right to the dividend.

This strategy requires a dividend event feed, which the Intrinio adapter provides. You therefore need an Intrinio API key to run it.

J.2. Implementation

The main artifacts needed for the Implementation of a new Strategy are described in *Chapter 4, Strategy Development*.

The Dividend Capture Strategy consist of the following artifacts:

/src/main/java/ch/algotrader/strategy/dividends/DividendCaptureService.java

The strategy service class providing the main methods invoked by the application on certain events.

/src/main/resources/conf-dividends.properties

Contains parameters used by the strategy (e.g. `tradableSecurities` and `dividendAdapter` and `orderQuantity` and `accountId`)

/src/main/resources/db/mysql/mysql-data.sql

Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

To start the Strategy please see the explanations in *Chapter 3, Starting AlgoTrader*.

J.3. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:
Git Clone

If you have used the Section 2.1, “Windows Installer” to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```
$ git clone https://gitlab.algotrader.com/general/examples.git
```

Import/open the project dividends

To start the strategy in live trading mode on a development workstation please execute the following steps:

Initialize the database

- load the db-samples script into the MySQL database: `/algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql`
- load the strategy specific script into the MySQL database: `/dividends/src/main/resources/db/mysql/mysql-data.sql`

Start the Strategy

- invoke the Run Configuration: `EmbeddedStarter-dividends`
- In the VM arguments, set your Intrinio API key in the `-Dintr.apiKey=XXX`

To start the strategy in live trading mode on a productive server please execute the following steps:

Copy docker compose file

Copy the following file to the server and make changes as needed:

```
https://gitlab.algotrader.com/general/examples/blob/master/dividend-capture/docker-compose.yml
```

Run docker compose

Invoke the following command inside the directory where the `docker-compose.yml` file is located:

```
docker-compose up -d
```

Note

Prior to starting the strategy for the very first time please start the AlgoTrader server by itself by executing the following command inside `/bootstrap/launch`. this will load the MySQL sample data

```
docker-compose up -d mysql ibgateway algotrader
```
Appendix K. Example Strategy "NLP"

K.1. Trading Idea

**Warning**

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it live.

Natural Language Processing, shortened as NLP, is a branch of artificial intelligence that uses the natural language to deal with the interaction between computers and humans.

The purpose of NLP is to read, understand and analyse the human languages to make is valuable.

The idea of the strategy is to receive tweets about the chosen instrument and basing on their average sentiment enter long or short position.

The sentiment is calculated based on Standford CoreNLP library.

K.2. Implementation

The main artifacts needed for the implementation of a new strategy are described in Chapter 4, Strategy Development.

The NLP Strategy consist of the following artifacts:

/src/main/java/ch/algotrader/strategy/nlp/NlpService.java
The strategy service class providing the main methods invoked by the application on certain events.

/src/main/java/ch/algotrader/strategy/nlp/TweetService.java
The service class connects to Twitter and provides tweets.

/src/main/java/ch/algotrader/strategy/nlp/ScoreService.java
The service class uses natural language processing to calculate the overall sentiment of a tweet.

/src/main/resources/conf-nlp.properties
Contains parameters used by the strategy (e.g. accountId, securityId, orderQuantity, windowTimeInMinutes, openPositionFactor, closePositionFactor, consumerKey and consumerSecret)

/src/main/resources/db/mysql/mysql-data.sql
Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.
To start the Strategy please see the explanations in *Chapter 3, Starting AlgoTrader*.  

**K.3. Installation & Startup**

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

**Git Clone**

If you have used the *Section 2.1, “Windows Installer”* to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```
git clone https://gitlab.algotrader.com/general/examples.git
```

**Import/open the project nlp**

To start the strategy in live trading mode on a development workstation please execute the following steps:

**Initialize the database**

- **load the db-samples script into the MySQL database**: `/algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql`
- **load the strategy specific script into the MySQL database**: `/nlp/src/main/resources/mysql/mysql-data.sql`

**Set the Twitter keys**


**Start the Strategy**

- Use the run configuration below to start the strategy: `EmbeddedStrategyStarter-nlp`
Appendix L. Example Strategy "Market Maker"

L.1. Trading Idea

**Warning**

The purpose of this Strategy is to demonstrate capabilities of AlgoTrader. Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it live.

One can express their vision how market would behave in the nearest future. Having this in mind bid/ask price could be defined in the way that buying and selling at this price would be profitable. This example shows how market making strategy could be implemented. The key difference with other strategies is that market maker always ready to buy and sell instrument, thus providing liquidity to the market.

The main purpose of this strategy is to show how to use historical market data, live market data and do simple order management. The strategy calculates moving average price for USDBTC (using coinApi Pro historical data and coinbase Pro for live market data) and calculates mid price. The strategy uses assumption that current market mid price would reverse to the mean value (moving average in this example). During trading some position can accumulate, in such a case the strategy would place a hedge order which would square the position. Some statically configured spread will apply, also there is a threshold based on which current open orders would be adjusted (i.e. replaced according to the current market conditions).

L.2. Implementation

The main artifacts needed for the implementation of a new strategy are described in *Chapter 4, Strategy Development*.

The market making strategy consist of the following main artifacts:

/src/main/java/ch/algotrader/strategy/marketmaker/MarketmakerService.java

The strategy service class providing the main methods invoked by the application on certain events.

/src/main/java/ch/algotrader/strategy/marketmaker/algo/Metrics.java

The metrics class is responsible for calculating order parameters (i.e. buy/sell/hedge) according to the current market conditions.

/conf/settings.csv

Config where the following parameters are specified:

securityDescription - security to trade,
maxUnits - max amount strategy can trade on each side for market making orders,

repricingThreshold - threshold after which orders will be resubmitted with updated prices - it prevents resubmitting orders with small price changes.

/src/main/resources/db/mysql/mysql-data.sql
Contains the MySQL database records. Needs to be imported into the database before running the strategy with the MySQL database.

L.3. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:

Git Clone

If you have used the Section 2.1, “Windows Installer” to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

        git clone https://gitlab.algotrader.com/general/examples.git

Import/open the project marketmaker

To start the strategy in live trading mode on a development workstation please execute the following steps:

Initialize the database

load the db-samples script into the MySQL database: /algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql

load the strategy specific script into the MySQL database: /marketmaker/src/main/resources/db/mysql/mysql-data.sql

To start the Strategy please see the explanations in Chapter 3, Starting AlgoTrader. Please note: first reference data should be loaded. The following run configurations should be used:

Coinbase (sandbox) Reference Data

/runConfigurations/ReferenceDataStarter-Coinbase-sandbox.run.xml

Coin API reference data

/runConfigurations/ReferenceDataStarter-CoinAPI.run.xml

Strategy

/runConfigurations/EmbeddedStrategyStarter-marketmaker.run.xml
Appendix M. Example Strategy "RSI"

M.1. Trading Idea

Warning

Do not use it with a Live Trading Account and real Money! The strategy might lead to large losses. Even when modifying or extending the Strategy use caution before trading it live.

This example strategy shows how market indicator (Relative Strength Index in this case) can be used to generate buy or sell signals and act accordingly. There is no much focus on qualitative side, i.e. RSI signal is used in a simple, trivial way. However one can see how indicator is constructed, feed with market events and how trades can be sent based on the indicator’s values.

There is a simple rule for position accumulation - it is allowed to sell only when there is no position or there is only long position, and it is allowed to buy only when being flat or short.

M.2. Implementation

The main artifacts needed for the implementation of a new strategy are described in Chapter 4, Strategy Development.

The RSI strategy consist of the following main artifacts:

/src/main/java/ch/algotrader/strategy/rsi/RSIService.java
The strategy service class providing the main methods invoked by the application on certain events.

/src/main/resources/conf-rsi.properties
Config where the following parameters are specified:

- period - days to look back and calculate the RSI index,
- buyThreshold - index value, once value becomes below - treated as a buy signal,
- sellThreshold - index value, once value above - treated as a sell signal,
- securityId - id of the security to trade,
- quantity - order quantity.

M.3. Installation & Startup

To setup the strategy for back testing and live trading on a development workstation please execute the following steps:
Git Clone

If you have used the `Section 2.1, “Windows Installer”` to setup AlgoTrader, this project is already setup in the AlgoTrader IntelliJ IDEA and you can skip this step.

Perform a Git clone from the command line:

```bash
git clone https://gitlab.algotrader.com/general/examples.git
```

Import/open the project `rsi`

To start the strategy in live trading mode on a development workstation please execute the following steps:

Initialize the database

- load the `db-samples` script into the MySQL database: `/algotrader-conf/src/main/resources/db-samples/mysql/mysql-data.sql`

- load the strategy specific script into the MySQL database: `/rsi/src/main/resources/db/mysql/mysql-data.sql`

To start the Strategy please see the explanations in `Chapter 3, Starting AlgoTrader`. 
Appendix N. Example strategy "EMA" in Python

N.1. Description

This strategy effectively replicates the Java based EMA strategy (Appendix E, Example Strategy "EMA"). It is written in Python and integrates with the platform using the AlgoTrader Python Interface. The AlgoTrader Python Interface allows writing strategies in the same way as in Java, with access to the same event handler methods (e.g. onInit, onBar, onTick, onOrderStatus, etc.) and services (OrderService for order placement, AccountService for retrieval of account balances and initiation of withdrawals, etc.).

For installation instructions see the original strategy description: Section E.3, “Installation & Startup”. You may use only a subset of historical data from the EURUSD csv file (by e.g. only keeping say 500 first rows) to be able to run backtests during development faster.

The strategy can be written and run in both Python 2 and 3. The AlgoTrader Python Interface works with both Python versions and has been tested with Python 2.7 and 3.7. Both Python versions and the AlgoTrader Python Interface should already be installed in your environment if you are using an AWS instance running an AlgoTrader trial image. If you have a different set up, you need to install the Python package (Section 2.2.3, “Python installation”).

Running ema-python/ema-python-strategy.py in Python starts the strategy execution. The AlgoTrader Python Interface the strategy uses waits for AlgoTrader to be started if it is not started already.

AlgoTrader (running in Java) can be started using the EmbeddedStrategyStarter (for live trading, or live trading against an exchange simulator, Section 5.1, “Exchange Simulator”) or using the SimulationStarter for strategy backtesting on historical data (see Section 3.1, “Simulation Mode”). The spring profile pythonIntegration needs to be activated with those. You can use the Run Configurations EmbeddedStrategyStarter-ema-python and SimulationStarter-simulate-ema-python in /ema-python folder for that.

N.1.1. Implementation

Detailed information on implementing strategies in Python can be found in Section 4.5, “Strategy Development in Python”.

The strategy implementation extends the StrategyService class from the AlgoTrader Python Interface (algotrader_com package). Overridden methods (on_bar, on_init, etc.) contain the same logic as the overridden methods (onBar, onInit, etc.) in the Java version of the EMAService (Appendix E, Example Strategy "EMA"). For example the overridden on_bar method implements the EMA indicator calculation and the evaluation of strategy rules based on the crossover of 2 EMA indicators with different lengths.
connect_to_algotrader
This function is used to pass a strategy implementation (a class extending `StrategyService`) to the AlgoTrader Python Interface.

wait_for_algotrader_to_disconnect
This function is used to prevent the Python script from finishing.

_numpy_ewma_vectorized_v2
This function calculates the exponential moving averages. Note that using the popular Pandas library functionality would have a negative impact on performance.

The Python EMA strategy displays the profit evolution using the `matplotlib` library after the backtest finishes.

Figure N.1. Python Profit Evolution
Appendix O. Example strategy
"BreakOut" in Python

O.1. Description

This strategy effectively replicates the Java and Esper based BreakOut strategy (Appendix A, Example Strategy "BreakOut"). It is written in Python and integrates with the platform using the AlgoTrader Python Interface. The AlgoTrader Python Interface allows writing strategies in the same way as in Java, with access to the same event handler methods (e.g., onInit, onBar, onTick, onOrderStatus, etc.) and services (OrderService for order placement, AccountService for retrieval of account balances and initiation of withdrawals, etc.).

For installation instructions, see the original strategy description: Section A.4, “Installation & Startup”. You may use only a subset of historical data from the EURUSD csv file (by e.g. only keeping say 500 first rows) to be able to run backtests during development faster.

The strategy can be written and run in both Python 2 and 3. The AlgoTrader Python Interface works with both Python versions and has been tested with Python 2.7 and 3.7. Both Python versions and the AlgoTrader Python Interface should already be installed in your environment if you are using an AWS instance running an AlgoTrader trial image. If you have a different set up, you need to install the Python package (Section 2.2.3, “Python installation”).

Running `breakOut-python/breakout-python-strategy.py` in Python starts the strategy execution. The AlgoTrader Python Interface the strategy uses waits for AlgoTrader to be started if it is not started already.

AlgoTrader (running in Java) can be started using the EmbeddedStrategyStarter (for live trading, or live trading against an exchange simulator, Section 5.1, “Exchange Simulator”) or using the SimulationStarter for strategy backtesting on historical data (see Section 3.1, “Simulation Mode”). The spring profile pythonIntegration needs to be activated with those. You can use the Run Configurations EmbeddedStrategyStarter-breakOut-python and SimulationStarter-simulate-breakOut-python in the /breakOut-python folder for that.

O.1.1. Implementation

Detailed information on implementing strategies in Python can be found in Section 4.5, “Strategy Development in Python”.

Blob

BreakoutStrategyService

The strategy implementation extends the StrategyService class from the AlgoTrader Python Interface (algotrader_com package). The method _entry contains the same logic as the entry method in Java version of the BreakOutService (Appendix A, Example Strategy "BreakOut").

blob

The business logic is written in Esper statements that are identical to the original Java based strategy. The only changes are in the ENTRY_LONG, ENTRY_SHORT statements that invoke not a subscriber on a
Java based Spring bean, but pass the values to the Python strategy's on_esper_subscriber_invocation method using @Subscriber(className='pythonStrategyService#onEsperSubscriberInvocation')

connect_to_algotrader
This function is used to pass a strategy implementation (a class extending StrategyService) to the AlgoTrader Python Interface.

wait_for_algotrader_to_disconnect
This function is used to prevent the Python script from finishing.
Appendix P. Example strategy "EMA" in Python via API

P.1. Description

This strategy effectively replicates the Java based EMA strategy (Appendix E, Example Strategy "EMA"). It is written in Python and integrates with the platform using the AlgoTrader API (REST and ActiveMQ based).

This is a demonstration of the concept that by utilizing AlgoTrader API it is possible to write strategies in languages other than Java. In order to run this strategy, AlgoTrader server must be started. Running `ema-python-via-api/ema-api.py` in Python afterwards starts the strategy execution. The strategy subscribes to market data on an ActiveMQ topic and submits orders via REST API.

Note that even though this approach allows one to access all AlgoTrader functionality and run a strategy against a running AlgoTrader server, a strategy written this way can't be used for backtesting (Section 3.1, “Simulation Mode”). You can however run this strategy in live trading mode against Section 5.1, “Exchange Simulator”. Backtesting Python strategies works if you write your strategy using AlgoTrader Python Interface, see Appendix N, Example strategy "EMA" in Python.

All the configurations settings for the example strategy are located inside `ema-api.py` file. Only an installed Python interpreter is required (the example `ema-api.py` script was tested with Python version 3.6.3)